Fig. 1

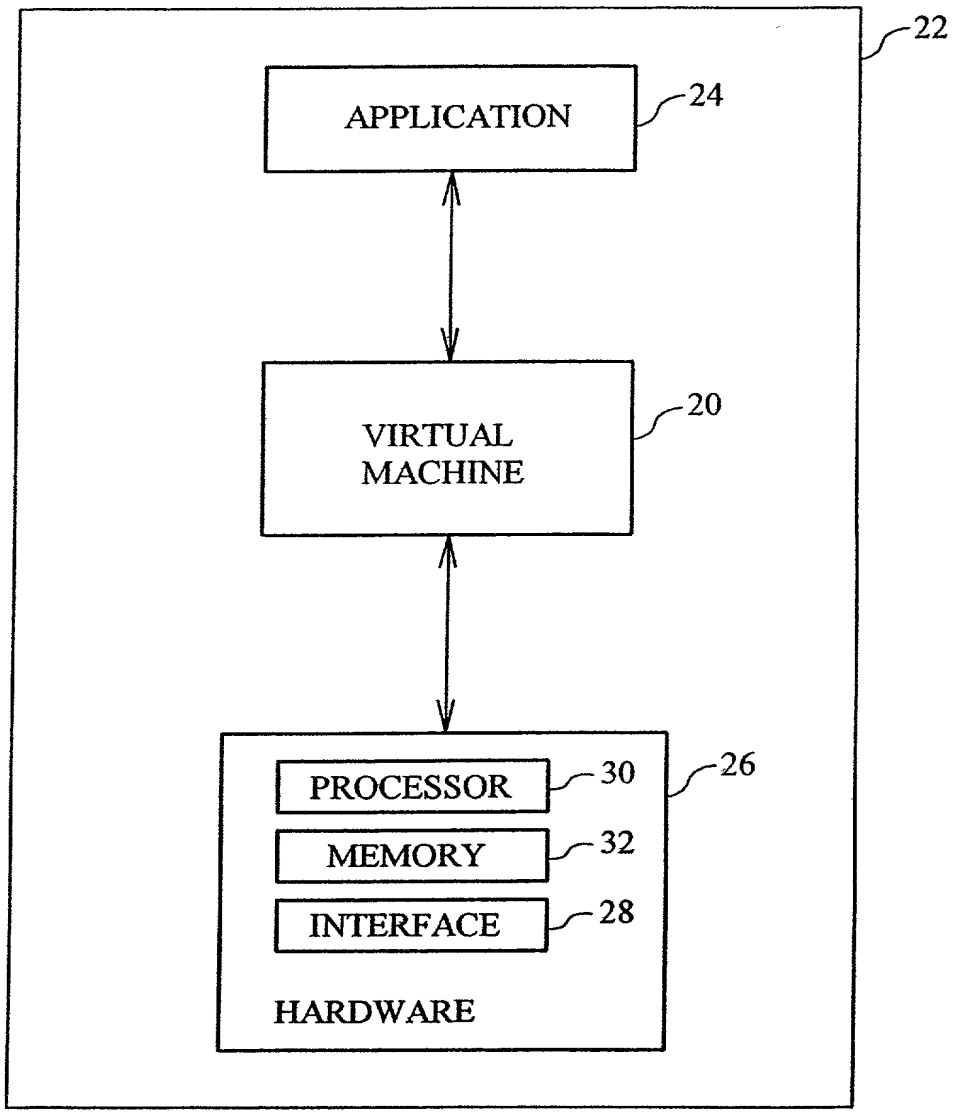

22
APPLICATION
24
VIRTUAL MACHINE
20
PROCESSOR
30
MEMORY
32
INTERFACE
28
HARDWARE
26

Fig. 1A

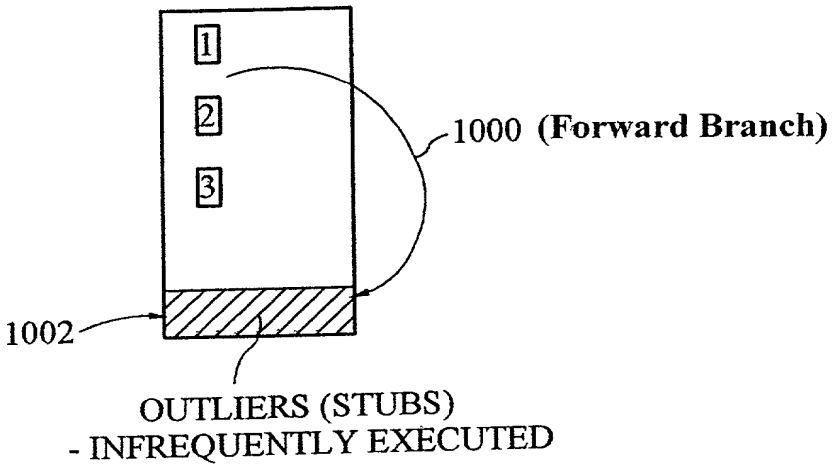
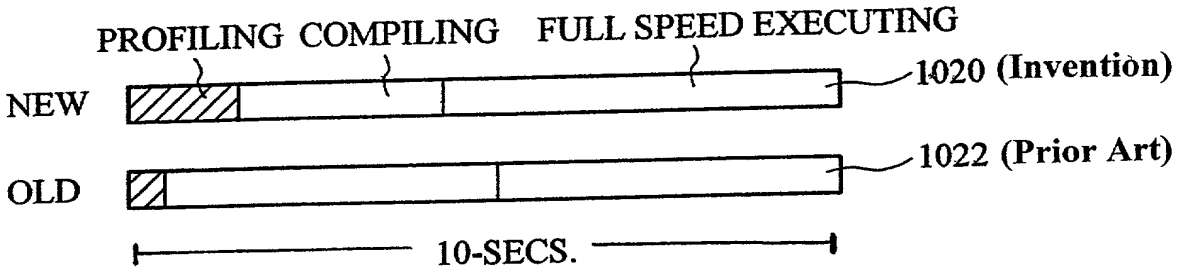

1
2
3
1000 (Forward Branch)
1002
OUTLIERS (STUBS)
- INFREQUENTLY EXECUTED

Fig. 1B


PROFILING
COMPILING
FULL SPEED EXECUTING
NEW
1020 (Invention)
OLD
1022 (Prior Art)
10-SECS.

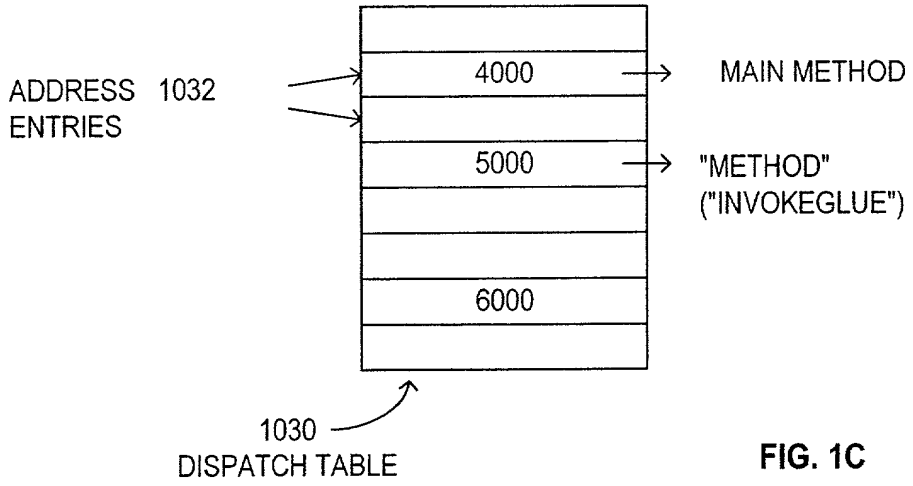

ADDRESS 1032
ENTRIES
4000
5000
6000
MAIN METHOD
"METHOD"
("INVOKEGLUE")
1030
DISPATCH TABLE

FIG. 1C

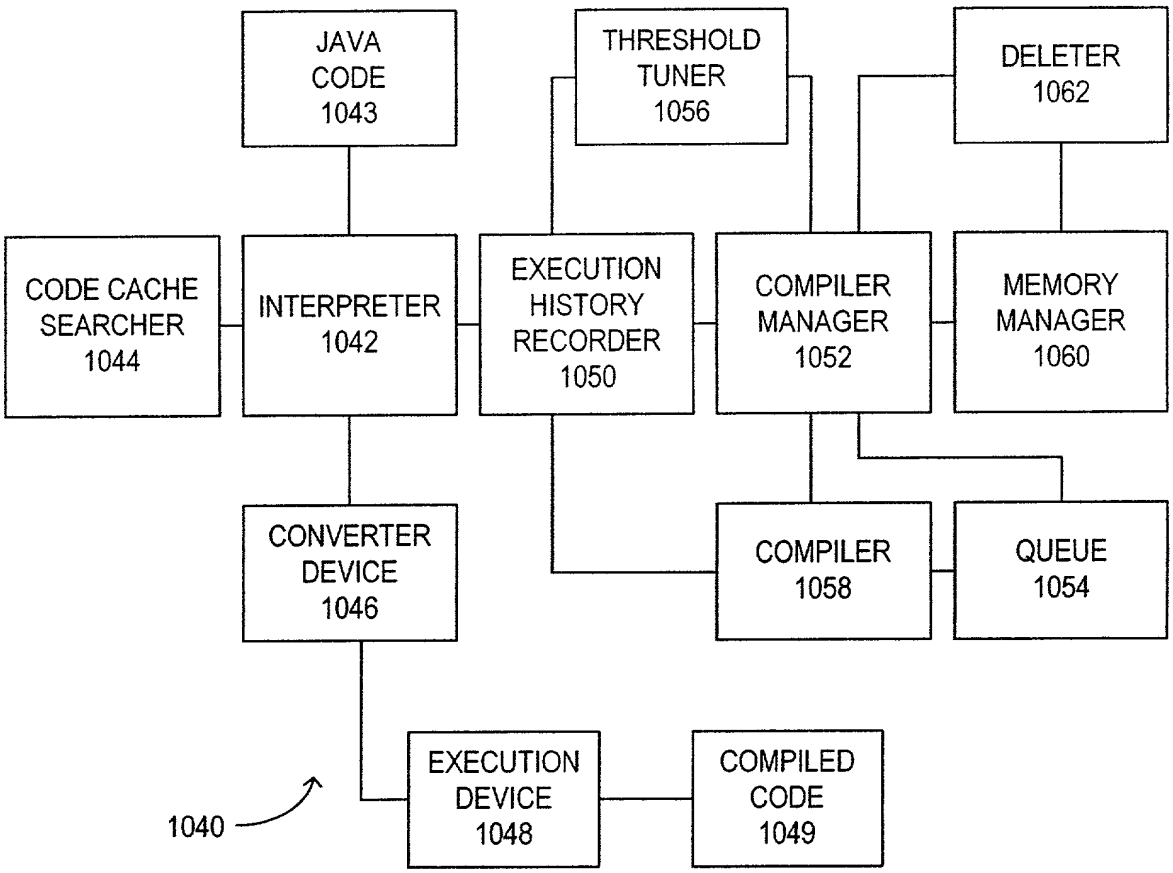

JAVA CODE 1043
THRESHOLD TUNER 1056
DELETER 1062
CODE CACHE SEARCHER 1044
INTERPRETER 1042
EXECUTION HISTORY RECORDER 1050
COMPILER MANAGER 1052
MEMORY MANAGER 1060
CONVERTER DEVICE 1046
COMPILER 1058
QUEUE 1054
EXECUTION DEVICE 1048
COMPILED CODE 1049
1040

FIG. 1D

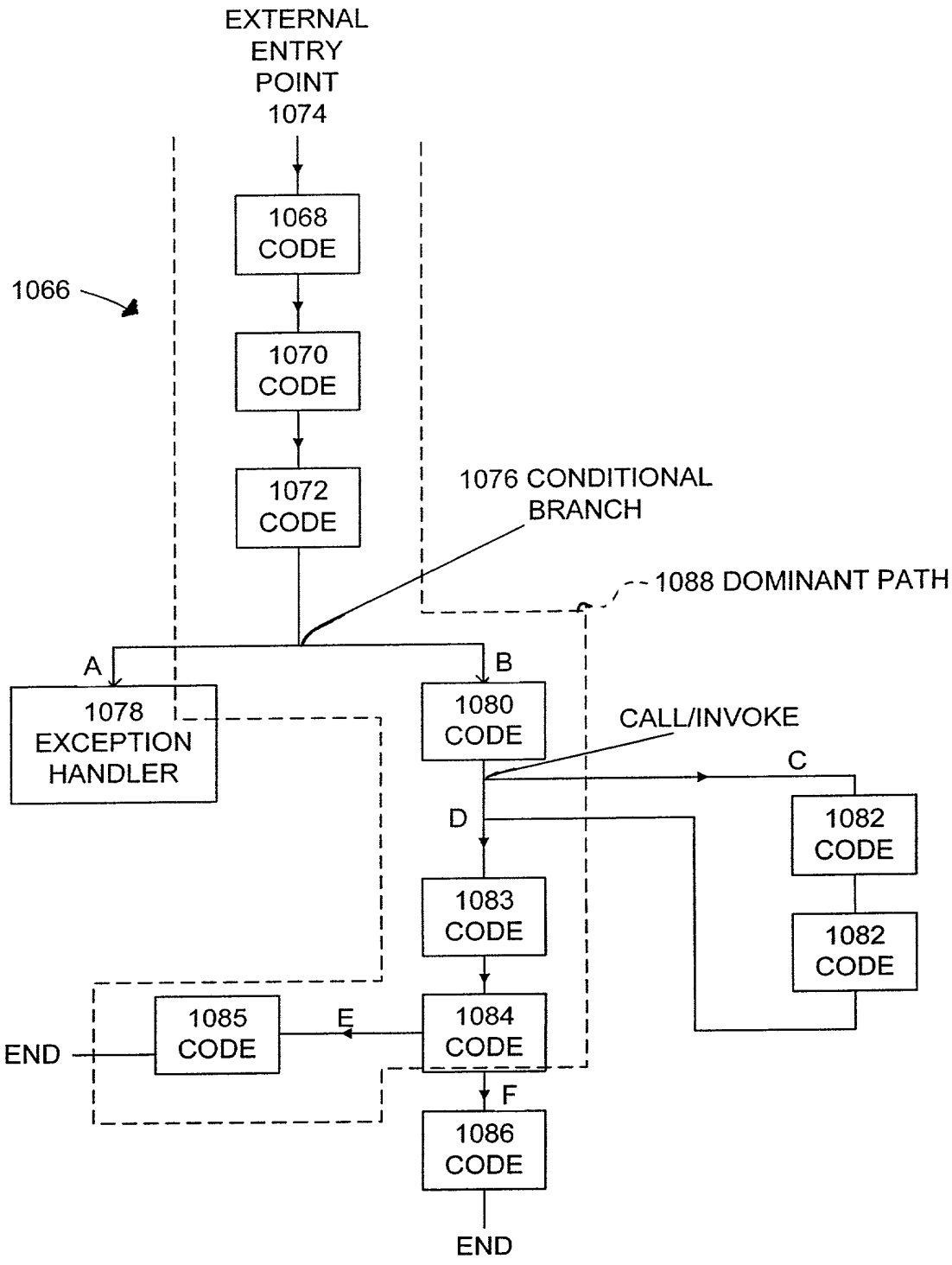

EXTERNAL ENTRY POINT 1074
1066
1068 CODE
1070 CODE
1072 CODE
1076 CONDITIONAL BRANCH
1088 DOMINANT PATH
A
B
1078 EXCEPTION HANDLER
1080 CODE
CALL/INVOKE
C
D
1082 CODE
1083 CODE
1082 CODE
1085 CODE
E
1084 CODE
END
F
1086 CODE
END

FIG. 1E

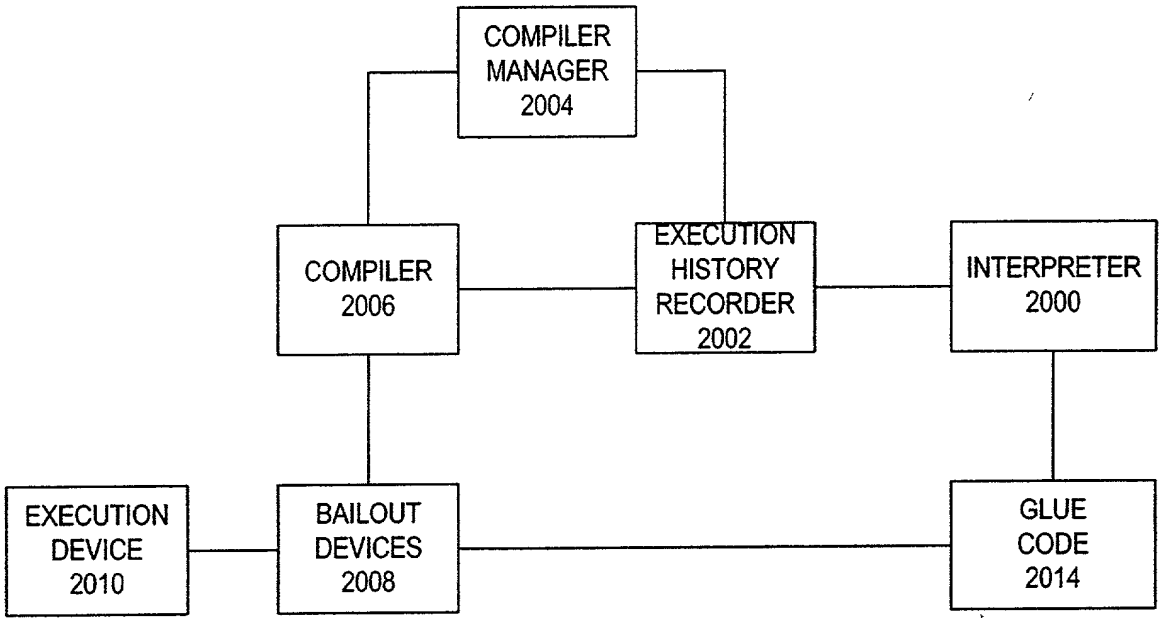


FIG. 2A

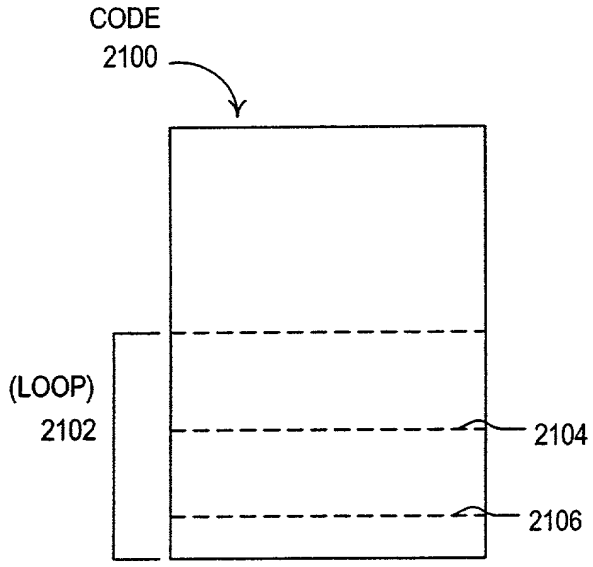


FIG. 2B

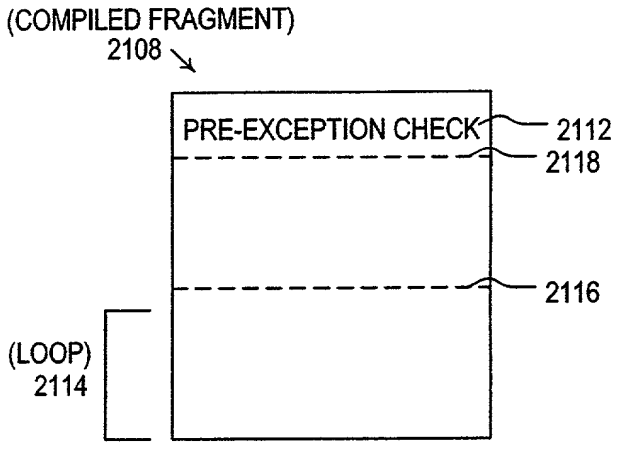


FIG. 2C

FIG. 3A

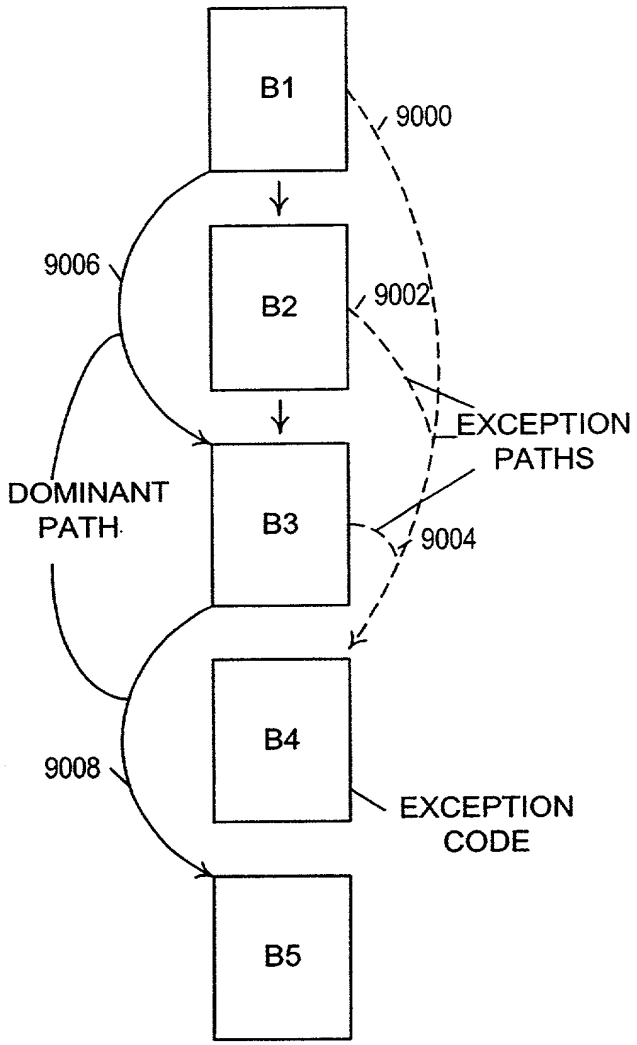


FIG. 3B

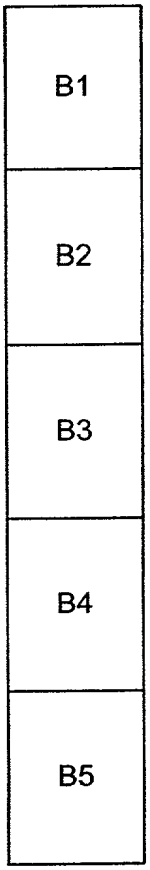
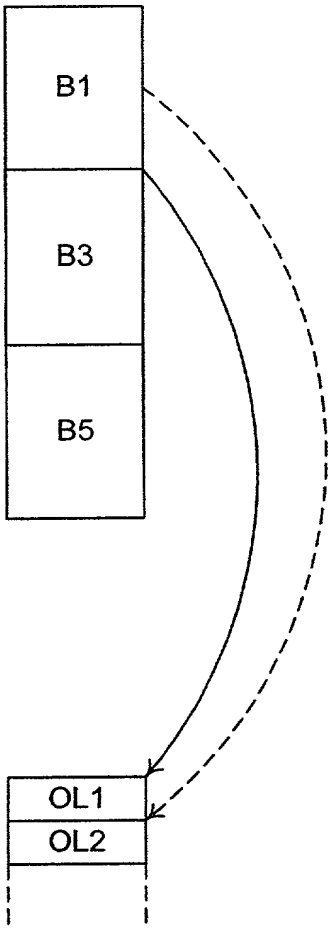


FIG. 3C



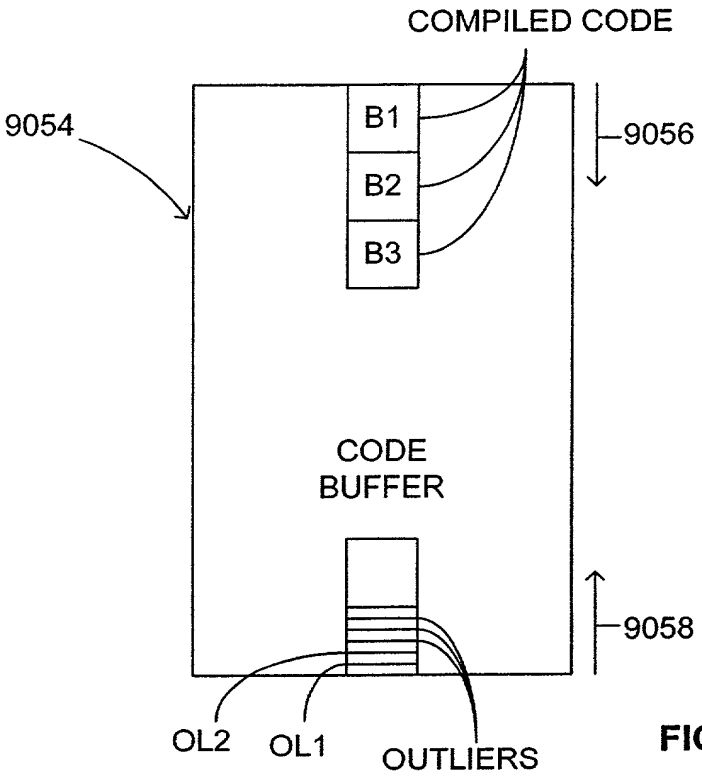
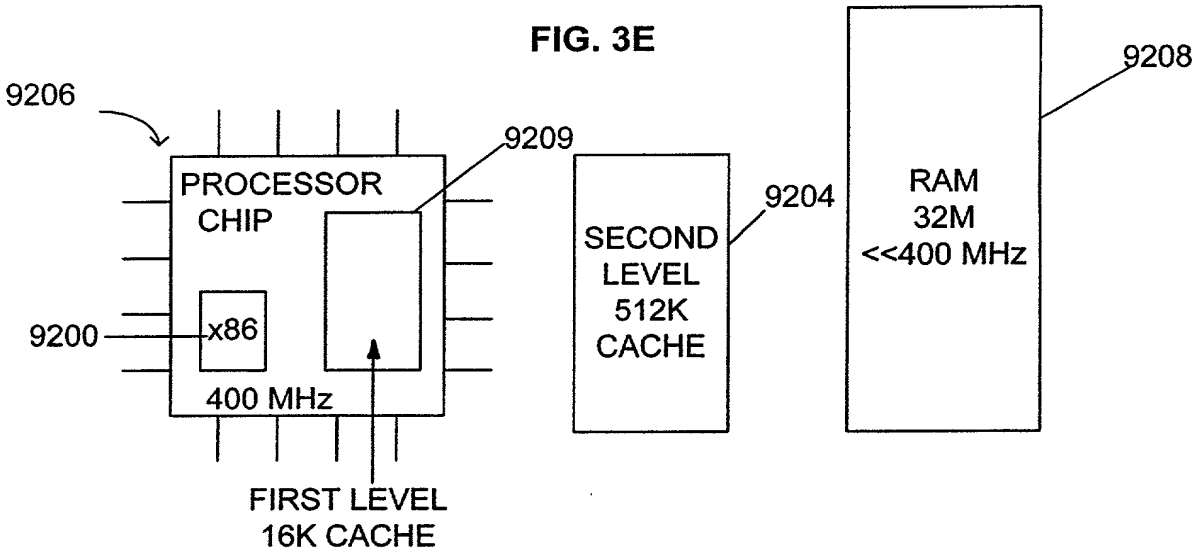

COMPILED CODE
9054
B1
B2
B3
9056
CODE
BUFFER
9058
OL2
OL1
OUTLIERS

FIG. 3D

FIG. 3E


9206
9209
PROCESSOR
CHIP
9200
x86
400 MHz
FIRST LEVEL
16K CACHE
SECOND
LEVEL
512K
CACHE
9204
RAM
32M
<<400 MHz
9208

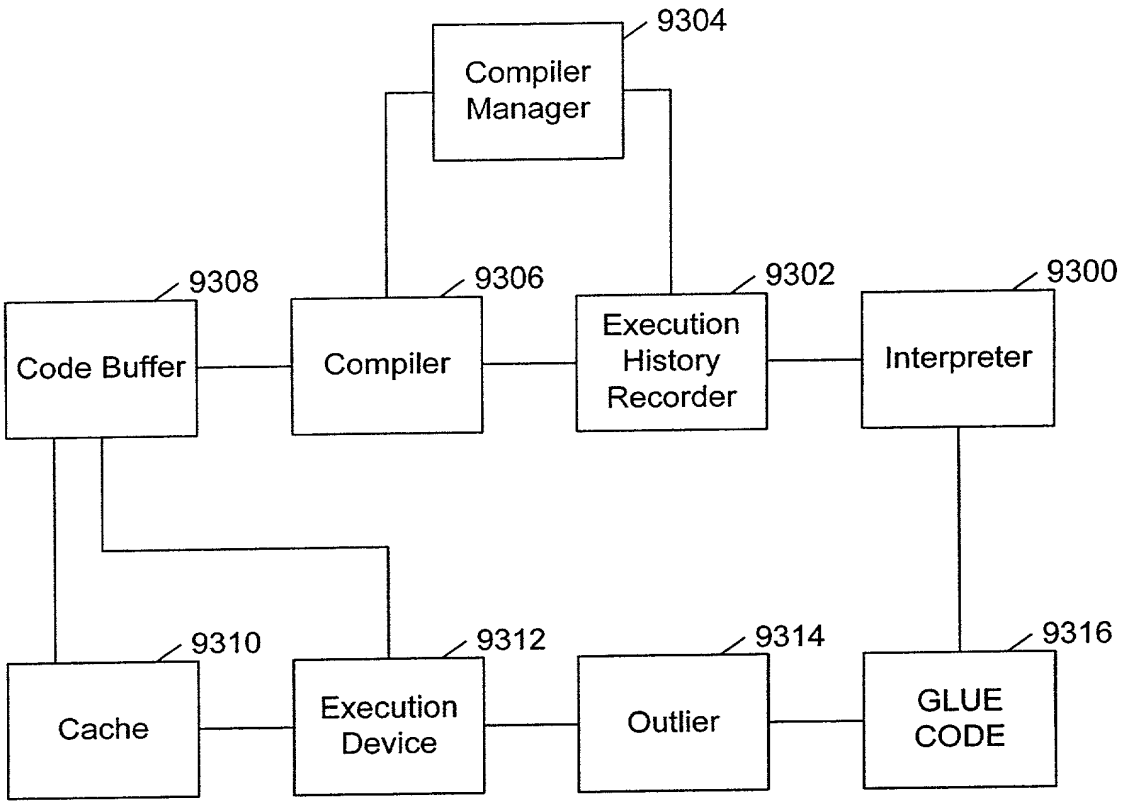

9304
Compiler Manager
9308
9306
9302
9300
Code Buffer
Compiler
Execution History Recorder
Interpreter
9310
9312
9314
9316
Cache
Execution Device
Outlier
GLUE CODE

FIG. 3F

Fig. 4A

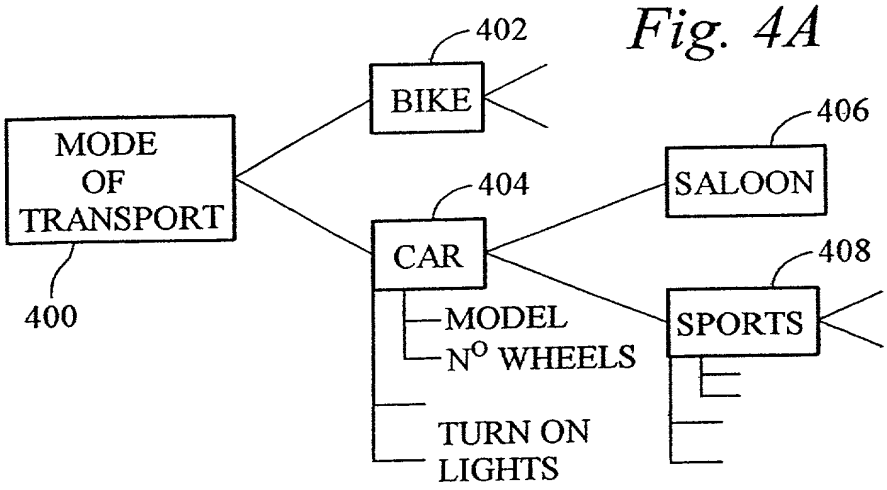


Fig. 4B

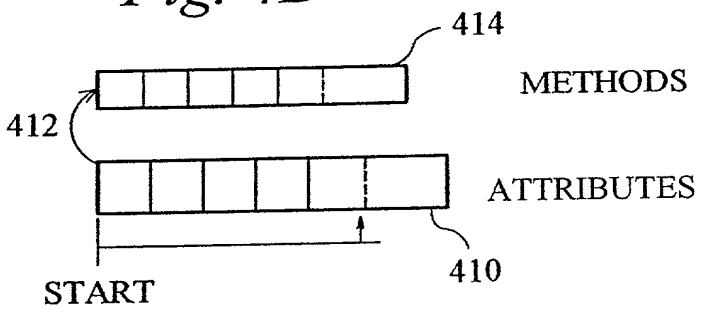


Fig. 4C

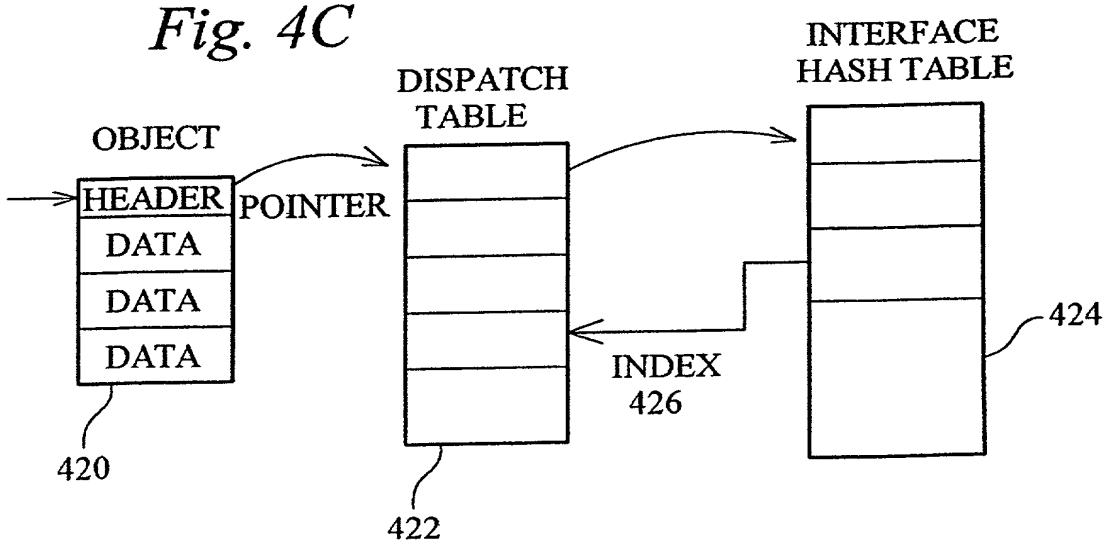
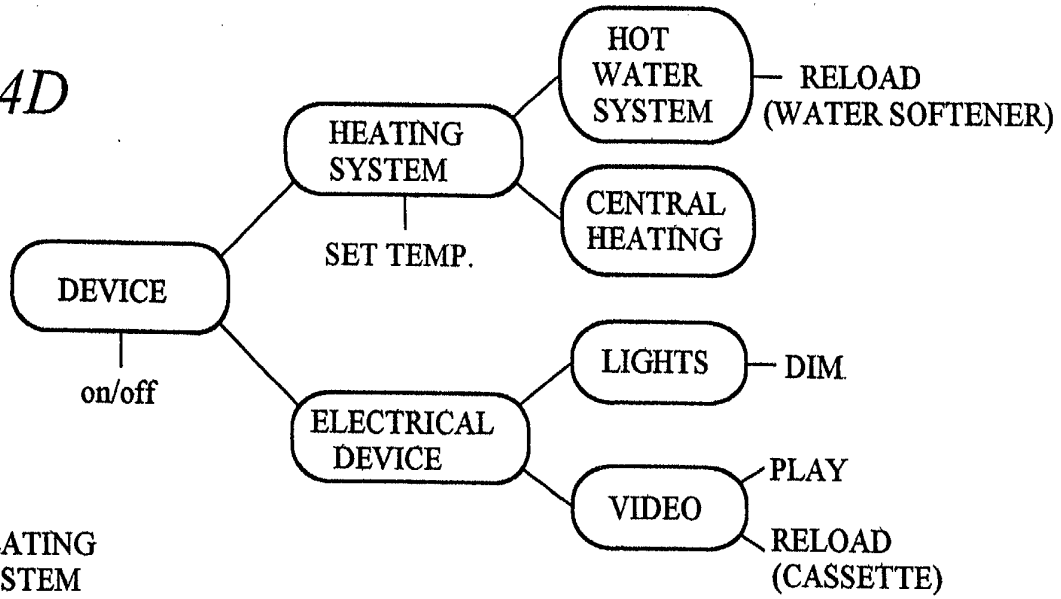


Fig. 4D



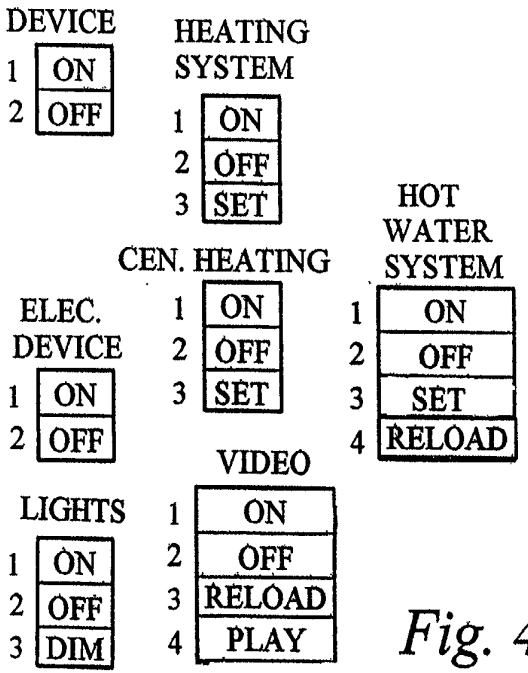
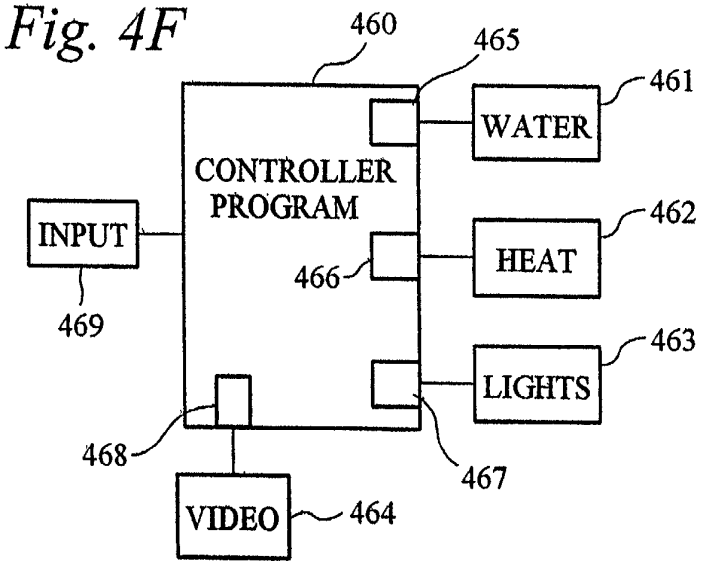


Fig. 4E

Fig. 4F



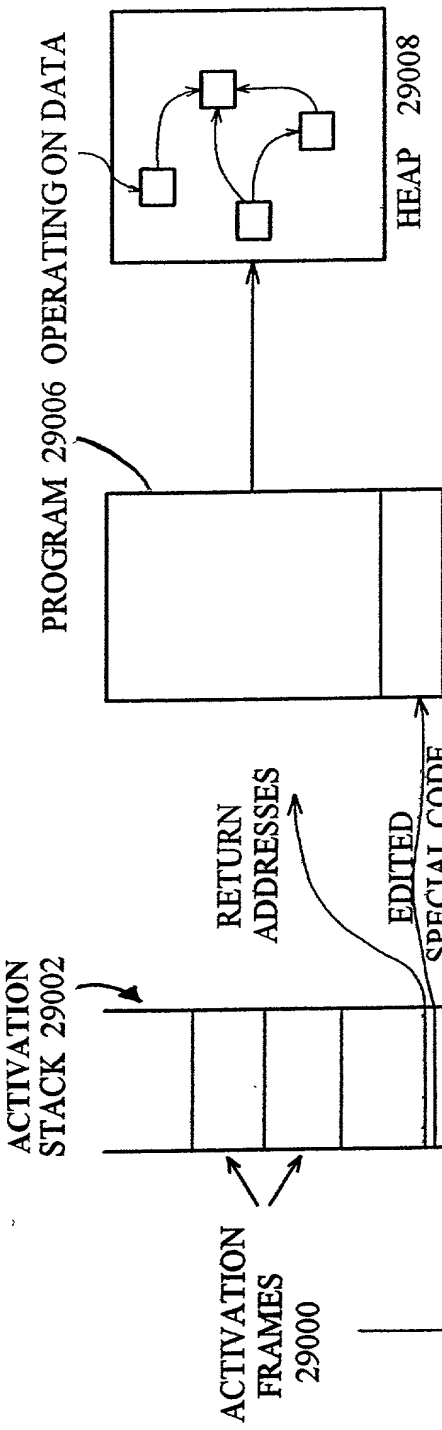

HEAP 29008
PROGRAM 29006 OPERATING ON DATA
RETURN ADDRESSES
EDITED SPECIAL CODE
ACTIVATION STACK 29002
ACTIVATION FRAMES 29000
29004
MOST RECENT ACTIVATION

Fig. 5A

Fig. 5B

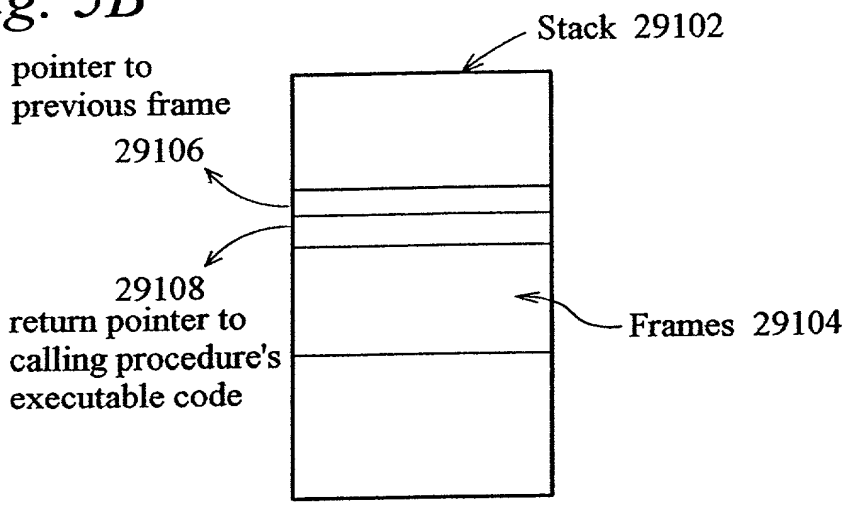


Fig. 5C

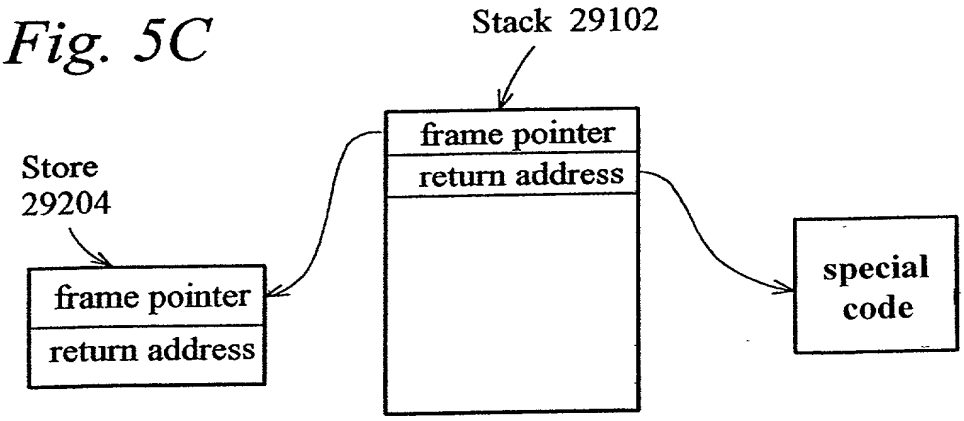


Fig. 5D

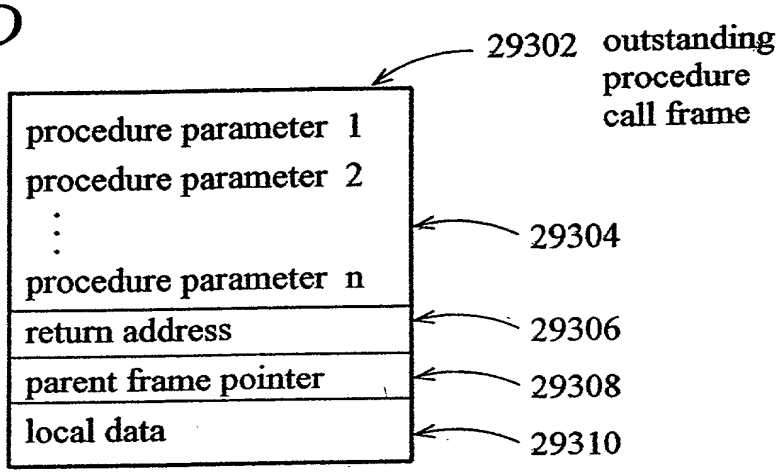


Fig. 5E

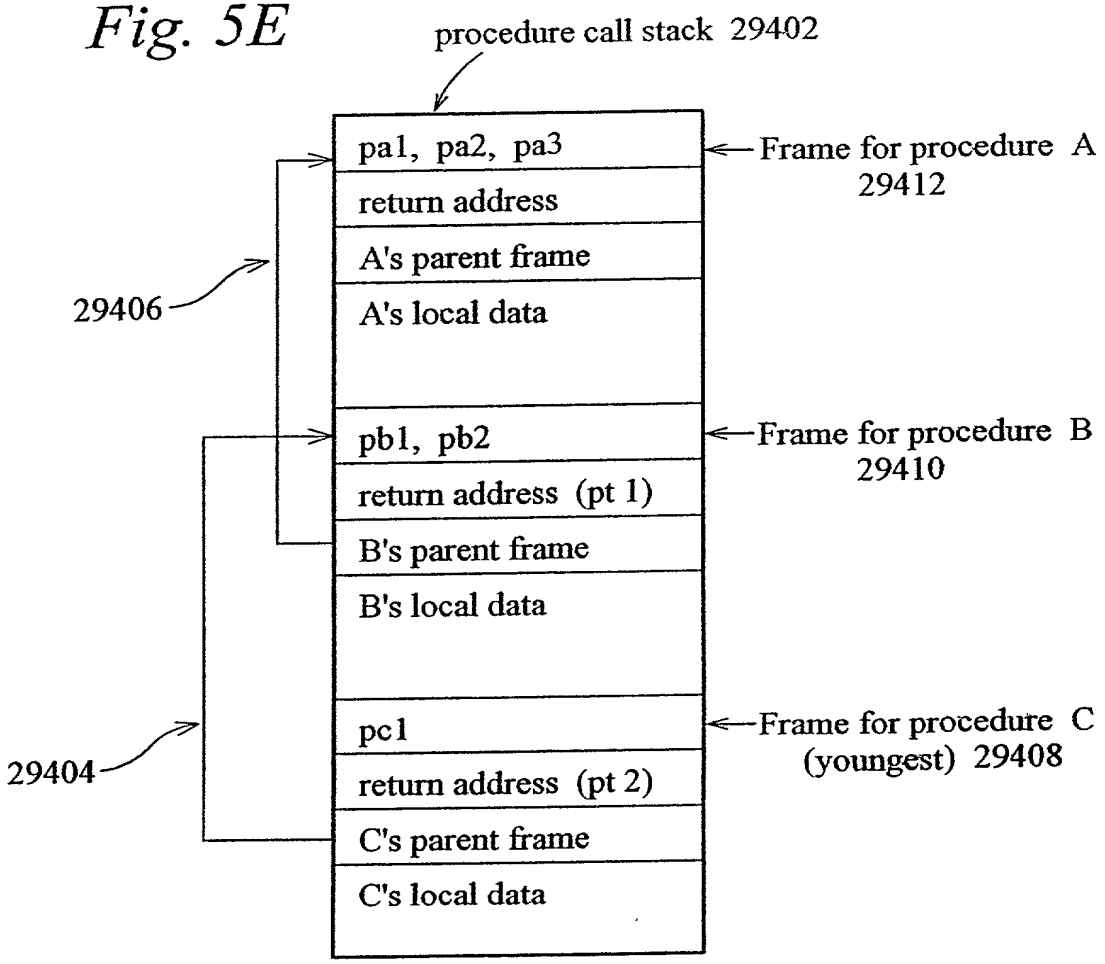
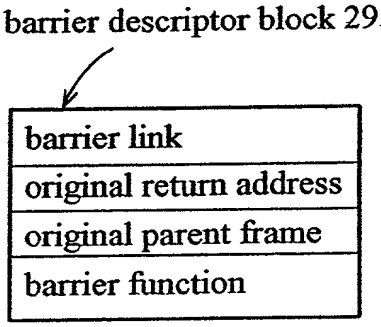

procedure call stack 29402
pa1, pa2, pa3
return address
A's parent frame
A's local data
Frame for procedure A 29412
29406
pb1, pb2
return address (pt 1)
B's parent frame
B's local data
Frame for procedure B 29410
pc1
return address (pt 2)
C's parent frame
C's local data
Frame for procedure C (youngest) 29408
29404

Fig. 5F


barrier descriptor block 29502
barrier link
original return address
original parent frame
barrier function

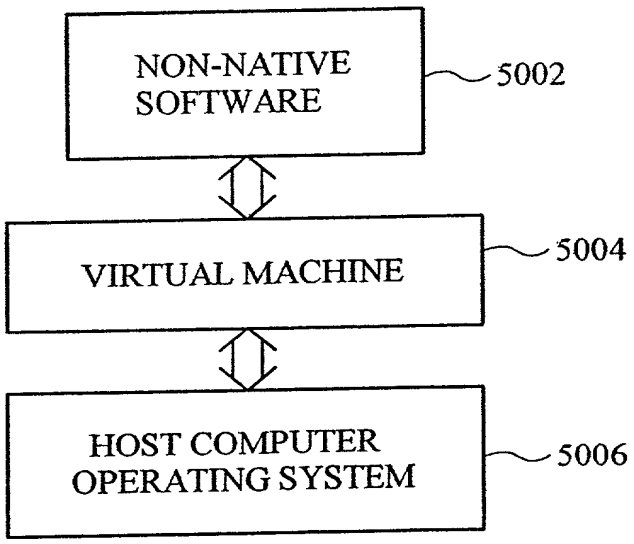

NON-NATIVE SOFTWARE
5002
VIRTUAL MACHINE
5004
HOST COMPUTER OPERATING SYSTEM
5006

Fig. 6A

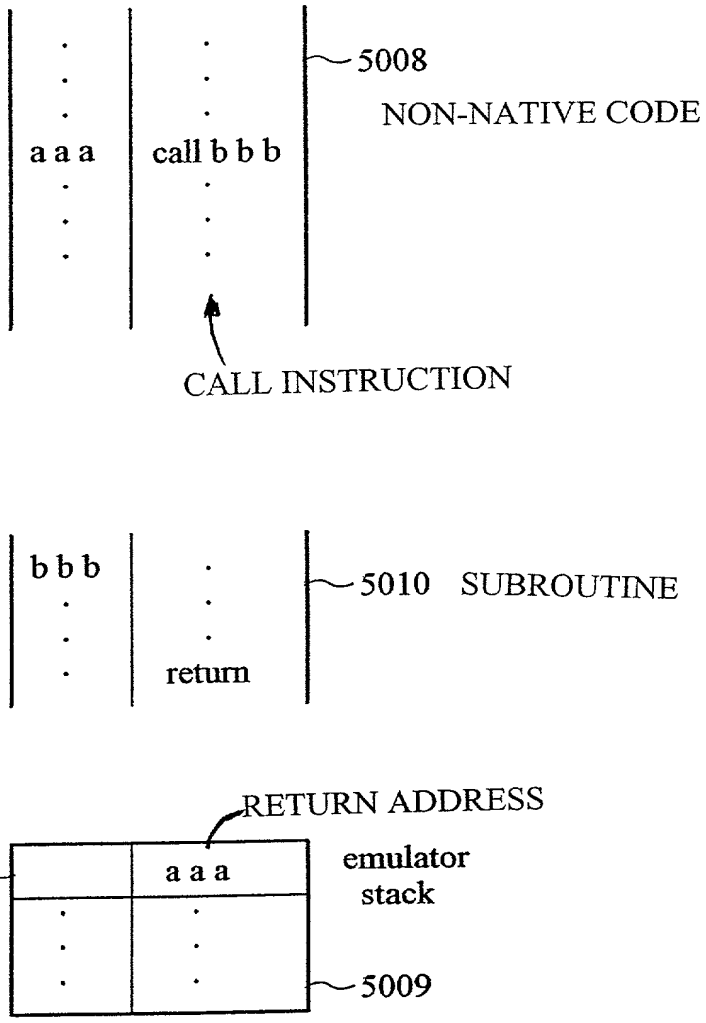

a a a
call b b b
5008
NON-NATIVE CODE
CALL INSTRUCTION
b b b
return
5010 SUBROUTINE
RETURN ADDRESS
register values
a a a
emulator stack
5009

Fig. 6B

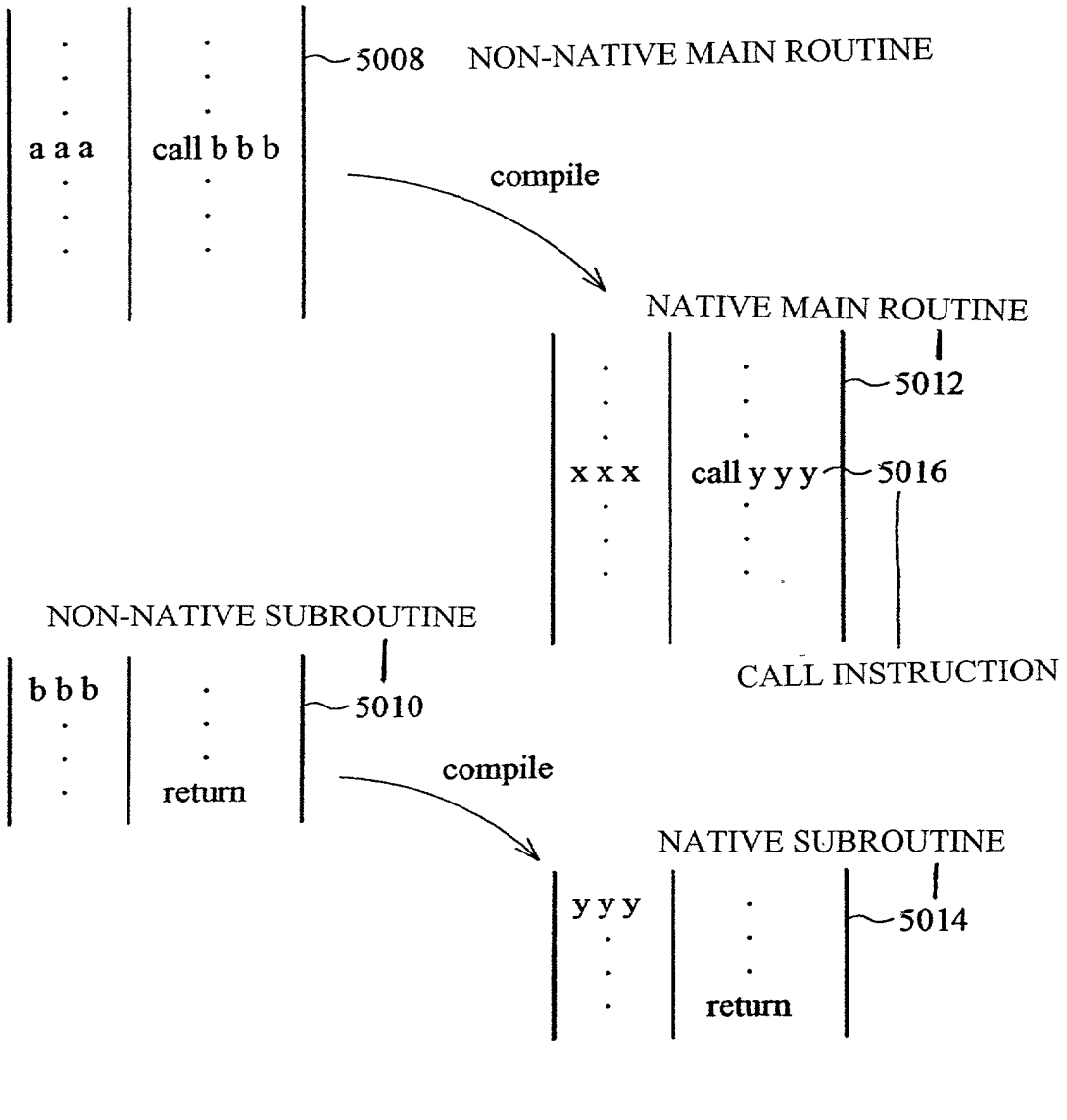

5008
NON-NATIVE MAIN ROUTINE
a a a
call b b b
compile
NATIVE MAIN ROUTINE
5012
x x x
call y y y
5016
CALL INSTRUCTION
NON-NATIVE SUBROUTINE
b b b
5010
return
compile
NATIVE SUBROUTINE
y y y
5014
return
RETURN ADDRESS
register values
x x x
host stack
5015

Fig. 6C

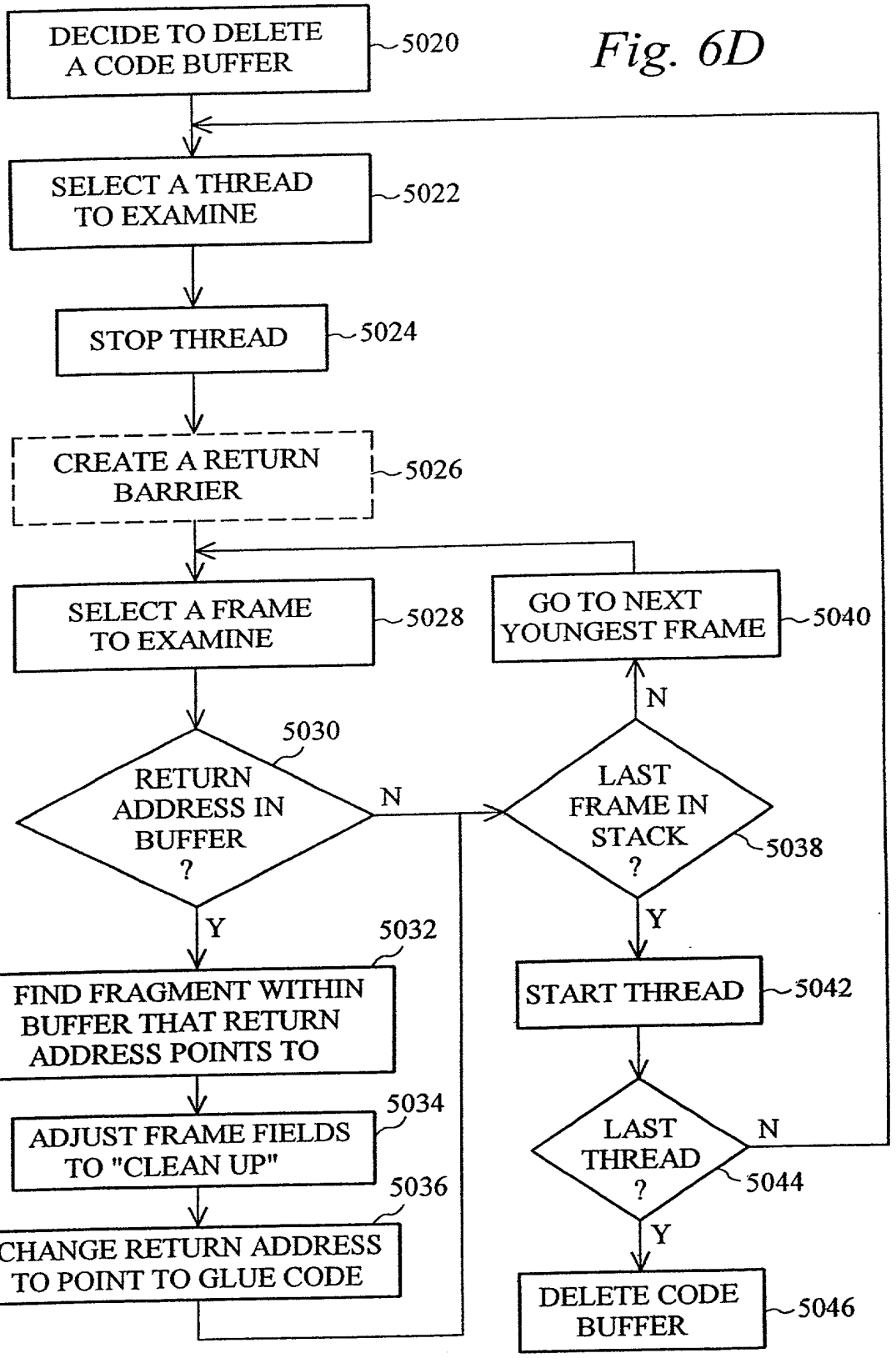

DECIDE TO DELETE A CODE BUFFER
5020
SELECT A THREAD TO EXAMINE
5022
STOP THREAD
5024
CREATE A RETURN BARRIER
5026
SELECT A FRAME TO EXAMINE
5028
5030
RETURN ADDRESS IN BUFFER ?
N
Y
5032
FIND FRAGMENT WITHIN BUFFER THAT RETURN ADDRESS POINTS TO
5034
ADJUST FRAME FIELDS TO "CLEAN UP"
5036
CHANGE RETURN ADDRESS TO POINT TO GLUE CODE
GO TO NEXT YOUNGEST FRAME
5040
N
LAST FRAME IN STACK ?
5038
Y
START THREAD
5042
LAST THREAD ?
5044
N
Y
DELETE CODE BUFFER
5046

Fig. 6D

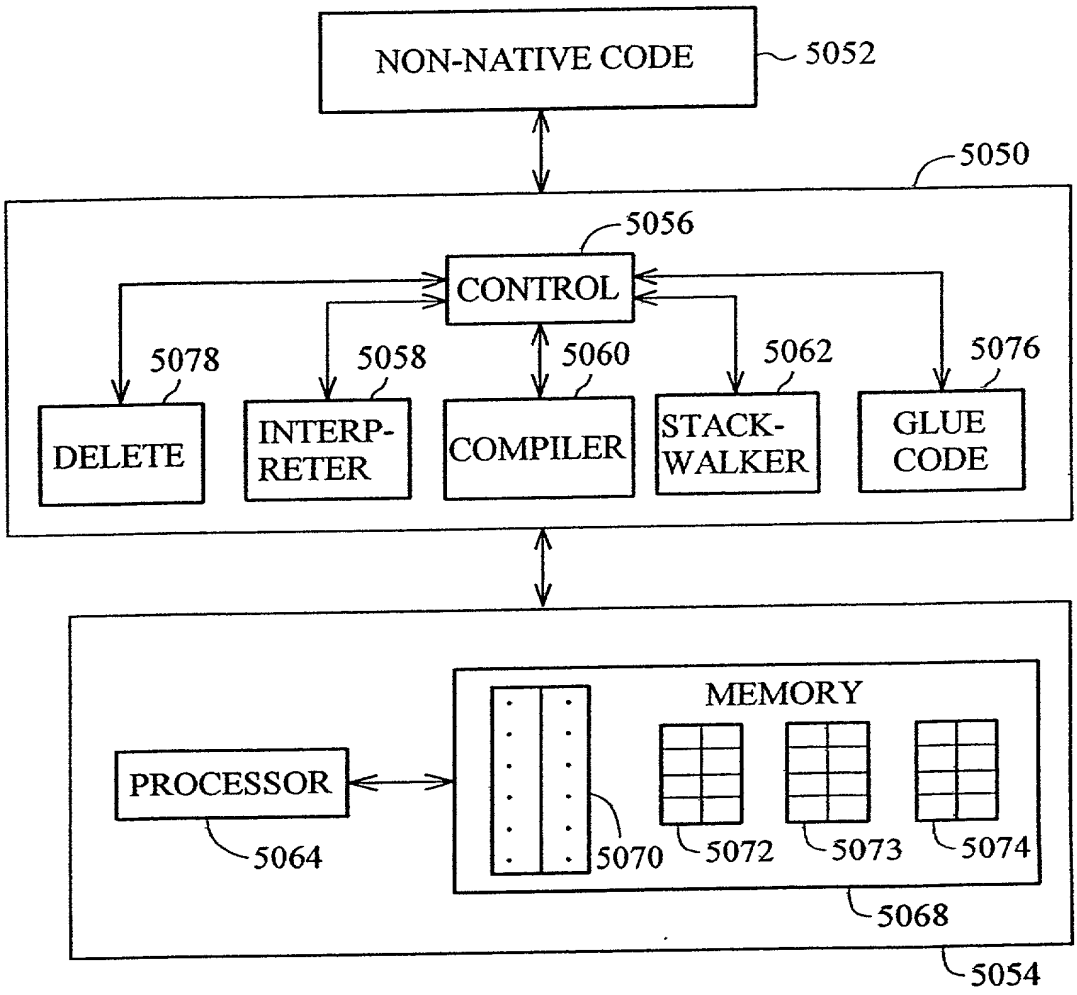

NON-NATIVE CODE
5052
5050
5056
CONTROL
5078
5058
5060
5062
5076
DELETE
INTERP-
RETER
COMPILER
STACK-
WALKER
GLUE
CODE
MEMORY
PROCESSOR
5064
5070
5072
5073
5074
5068
5054

Fig. 6E

Fig. 7A

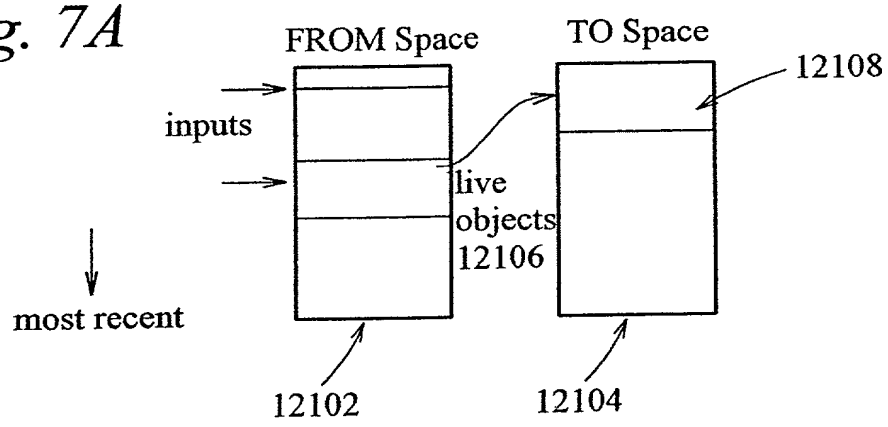


Fig. 7B

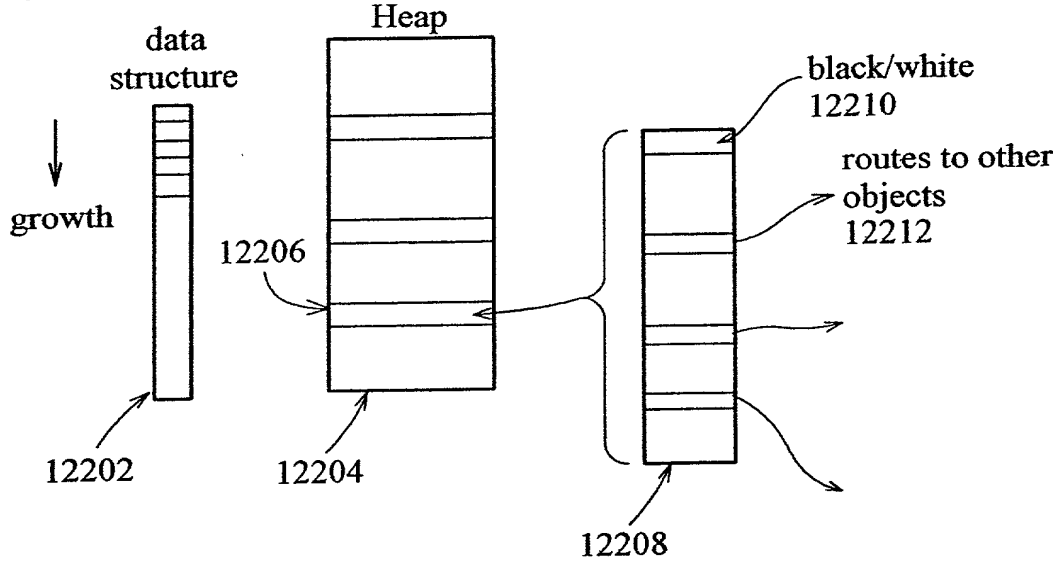


Fig. 7C

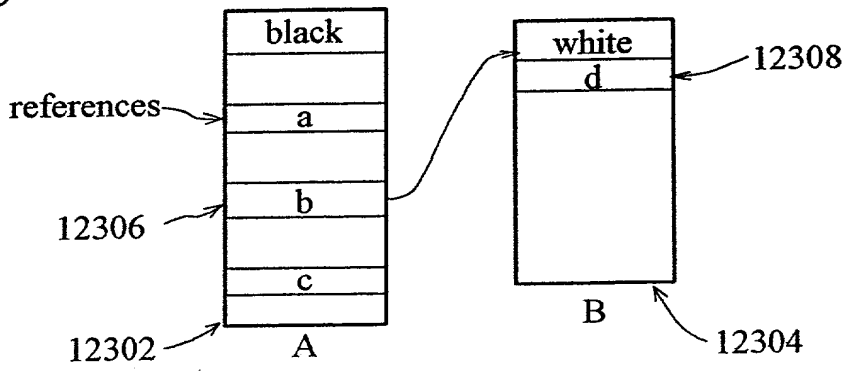


Fig. 7D

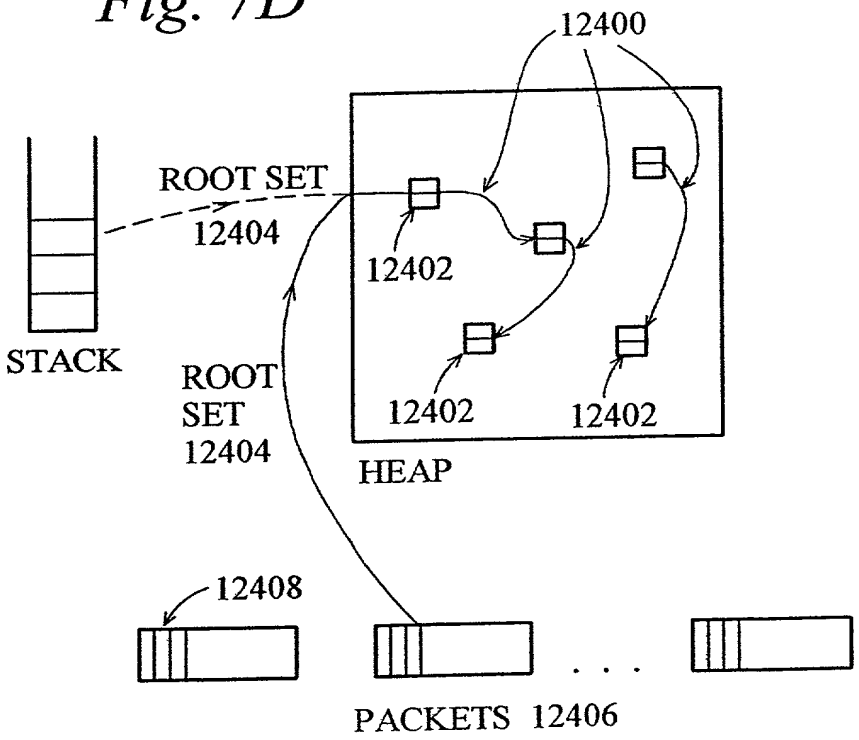

12400
ROOT SET
12404
12402
STACK
ROOT
SET
12404
12402
12402
HEAP
12408
PACKETS 12406

Fig. 7E

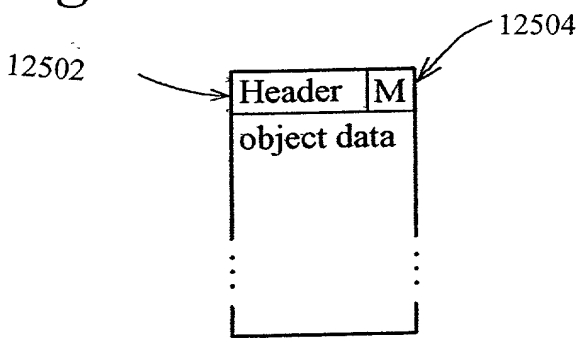


Fig. 7F

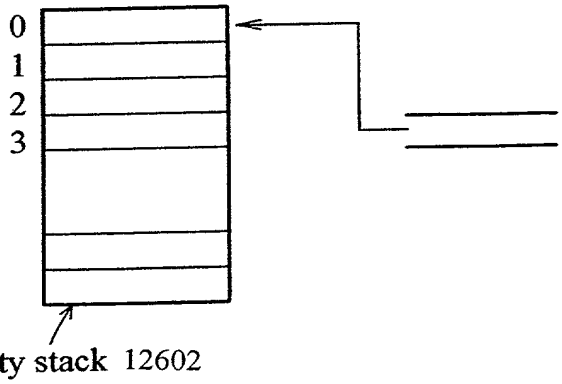


Fig. 7G

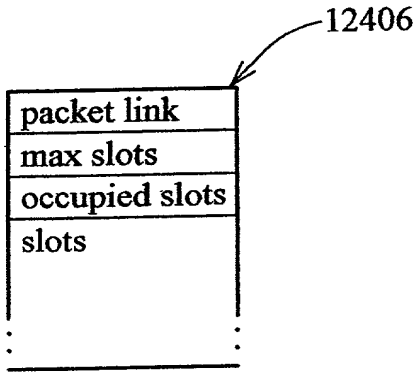
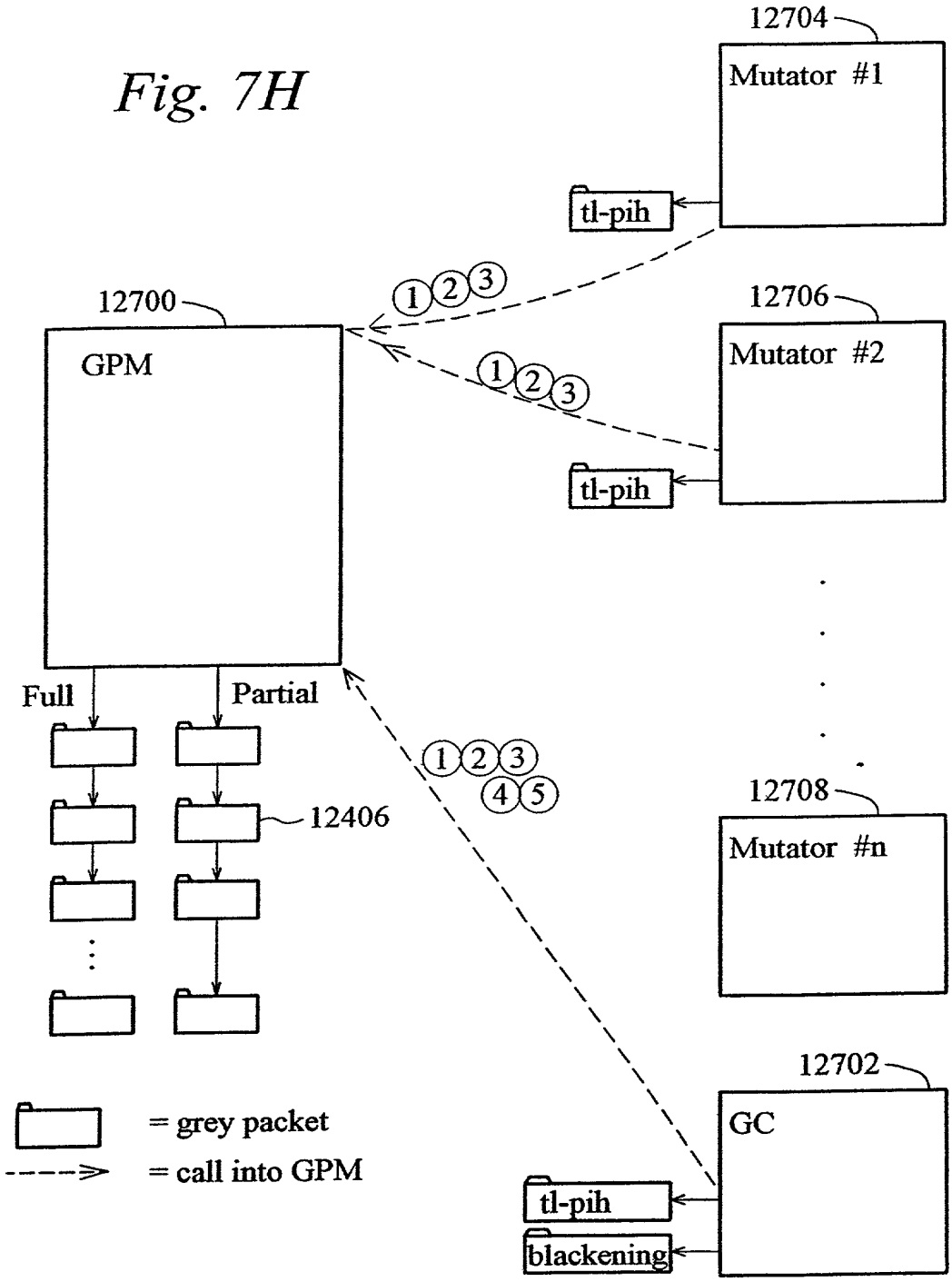


Fig. 7H


12704
Mutator #1
tl-pih
12700
GPM
1 2 3
12706
Mutator #2
1 2 3
tl-pih
Full
Partial
1 2 3
4 5
12406
12708
Mutator #n
12702
GC
= grey packet
= call into GPM
tl-pih
blackening

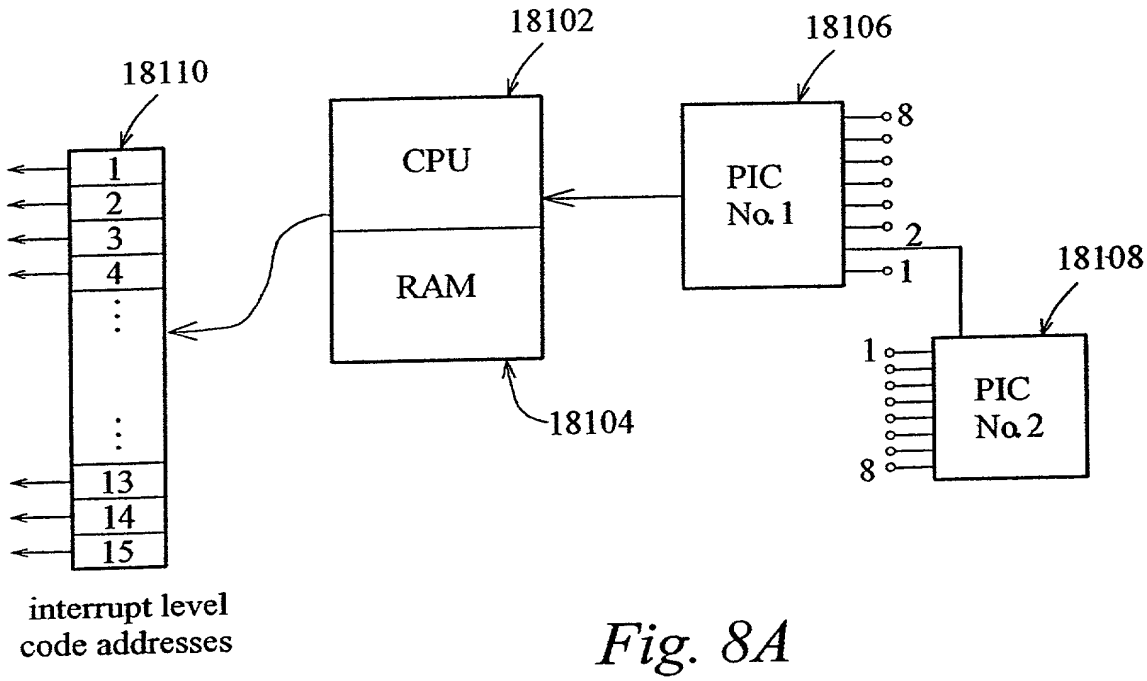

18102
18106
18110
CPU
RAM
PIC
No. 1
PIC
No. 2
18104
18108
1
2
3
4
13
14
15
8
2
1
1
8
interrupt level
code addresses

Fig. 8A

Sequence of Events for Various System Components *Fig. 8B*

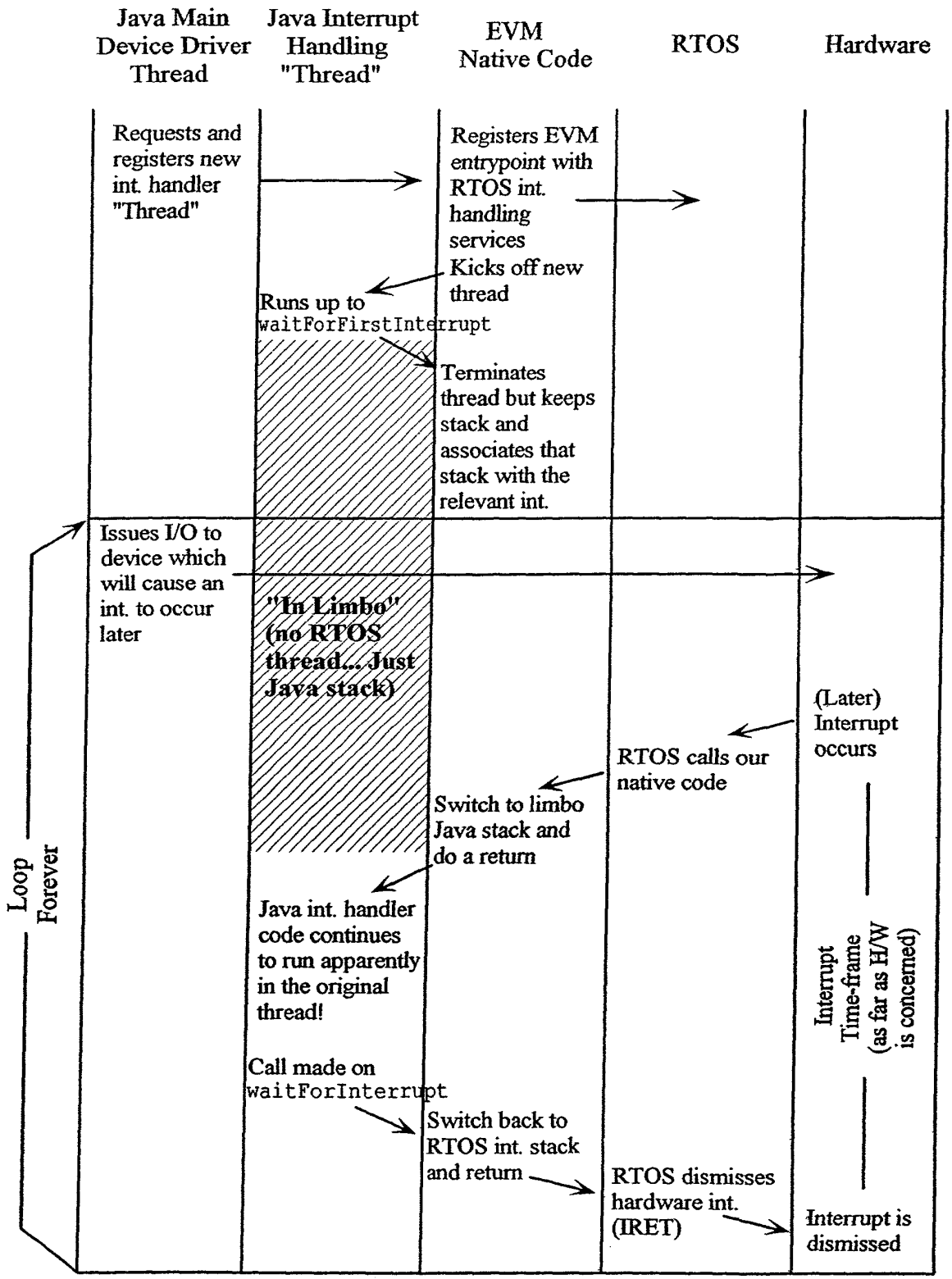


Fig. 8C-1

Pseudo-code of a Java Interrupt Handler

```
public void run ( )
{
        // The run method of an example interrupt handling thread

        // Wait for the first interrupt
        if  ( !waitForFirstInterrupt ( ) )
        {
                throw new RuntimeException  ( "Error waiting for 1st interrupt" ) ;
        }

        // We are now running at interrupt level!

        while  (true)
        {
                // Now handle the interrupt that just occurred (this involves
                // reading a device register)

                byte value = dev.readByte (DEV.DATA_REG);

                if  ( ( value & DEV.GOING_SYNC)  != 0)
                {
                        // We enter a sub-loop handling interrupts while in
                        // "synchronous mode"

                        boolean stillSync = true;
                        do
                        {
                                // Wait for the next interrupt to occur

                                waitForInterrupt ( ) ;

                                // Read the hardware data register

                                value = dev.readByte  (DEV.DATA_REG) ;
```

go to Fig. 8C-2

Fig. 8C-2

from Fig. 8C-1

```
                    // Decide if the value means that we are switching back
                    // to "async mode"

                    if ( (value & DEV.GOING_ASYNC) == 0)
                    {
                        // Handle "synchronous mode" interrupt here (just
                        // write the device data to non-interrupt code via
                        // the special channel)

                        specialChannel.write (value);
                    }
                    else
                    {
                        // Wait for the next interrupt to occur then return
                        // to the outer "async" loop

                        waitForInterrupt ( ) ;
                        stillSync = false;
                    }
                }
                while (stillSync)
            }

            // Handle "asynchronous mode" interrupts here (just write the
            // device data to non-interrupt code via the special channel)

            specialChannel.write  (value) ;

            waitForInterrupt ( ) ;
        }
    }
```

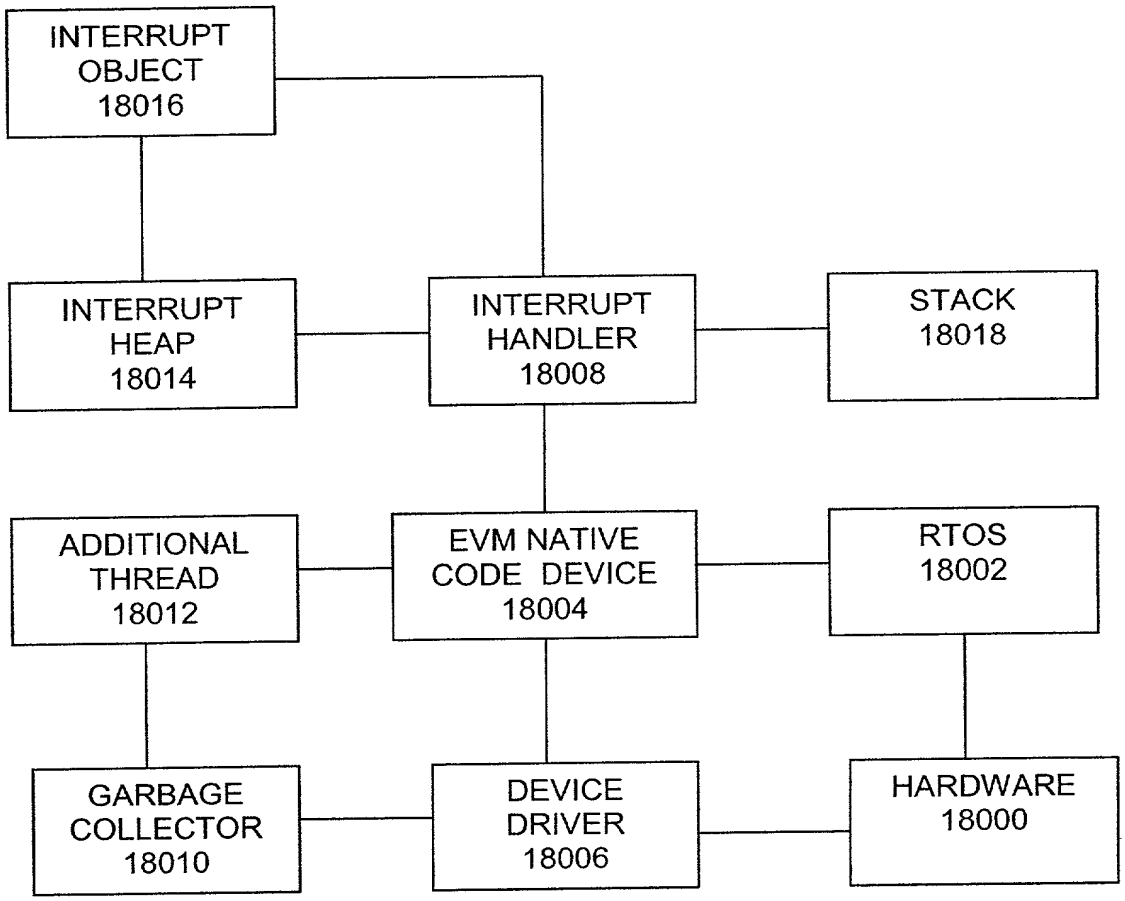
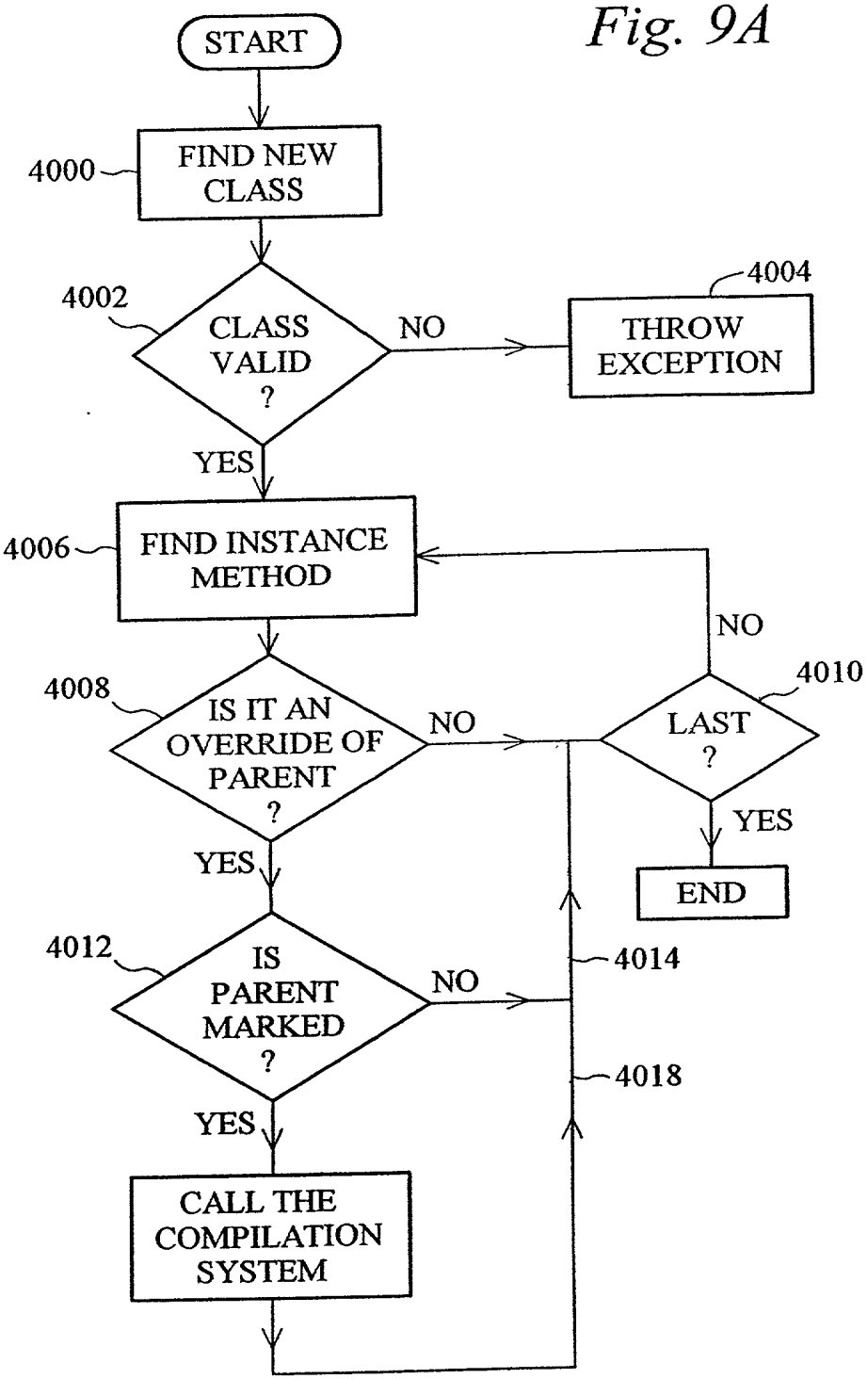

INTERRUPT OBJECT 18016
INTERRUPT HEAP 18014
INTERRUPT HANDLER 18008
STACK 18018
ADDITIONAL THREAD 18012
EVM NATIVE CODE DEVICE 18004
RTOS 18002
GARBAGE COLLECTOR 18010
DEVICE DRIVER 18006
HARDWARE 18000

FIG. 8D

Fig. 9A


START
4000
FIND NEW CLASS
4002
CLASS VALID ?
NO
4004
THROW EXCEPTION
YES
4006
FIND INSTANCE METHOD
NO
4008
IS IT AN OVERRIDE OF PARENT ?
NO
4010
LAST ?
YES
END
YES
4012
IS PARENT MARKED ?
NO
4014
4018
YES
CALL THE COMPILATION SYSTEM

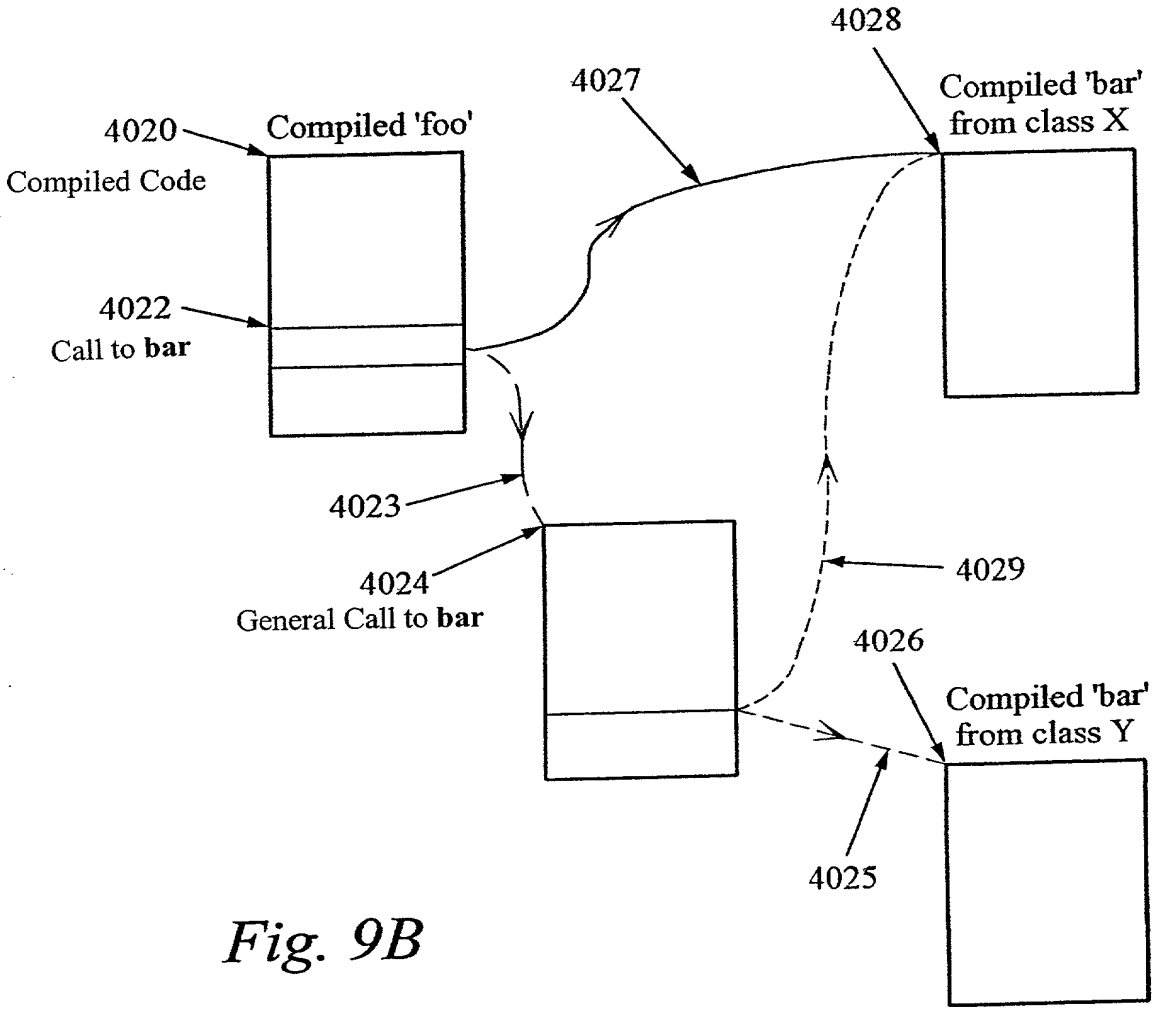

4028
4027
Compiled 'bar'
from class X
4020
Compiled 'foo'
Compiled Code
4022
Call to **bar**
4023
4024
General Call to **bar**
4029
4026
Compiled 'bar'
from class Y
4025

Fig. 9B

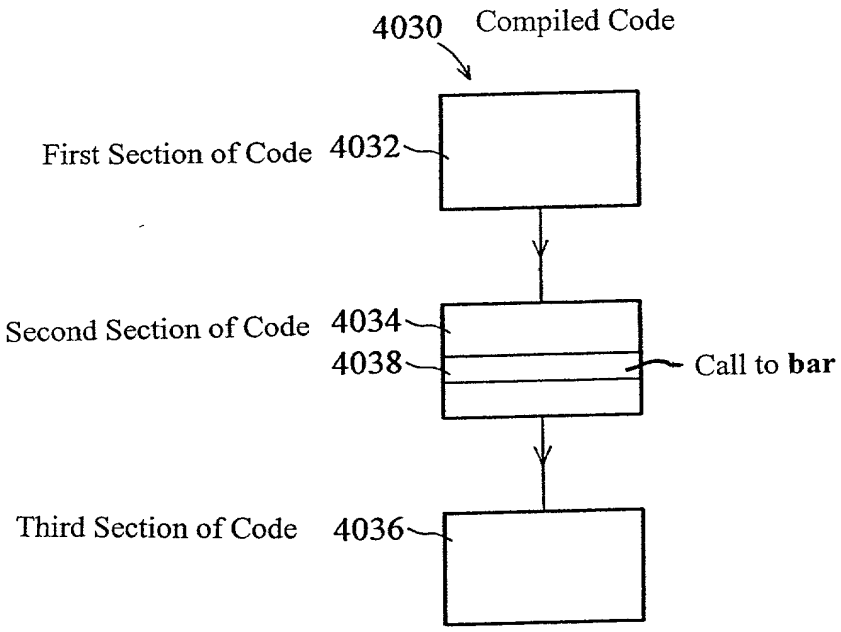

4030
Compiled Code
First Section of Code
4032
Second Section of Code
4034
4038
Call to bar
Third Section of Code
4036

Fig. 9C

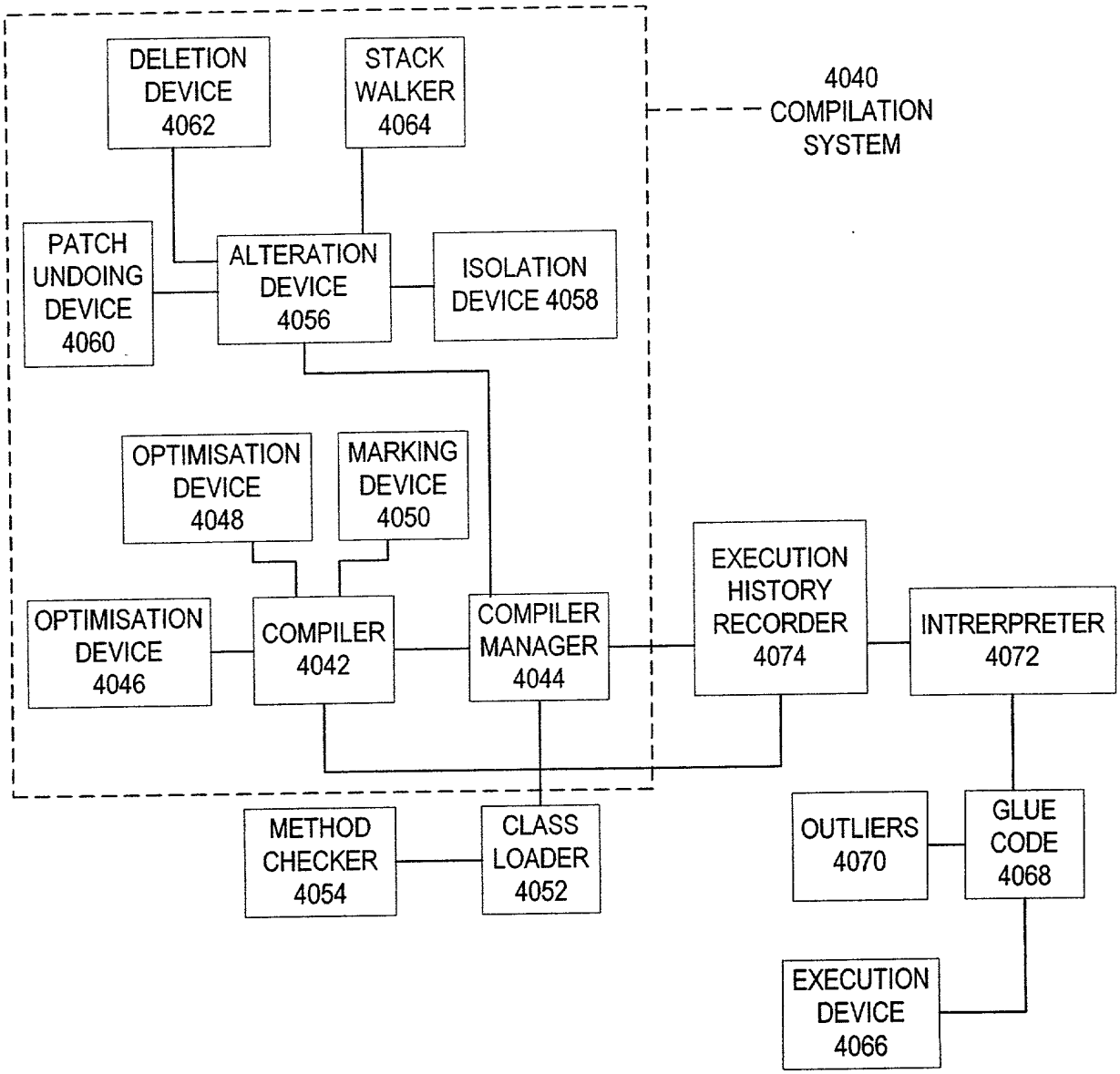

DELETION DEVICE 4062
STACK WALKER 4064
4040 COMPILATION SYSTEM
PATCH UNDOING DEVICE 4060
ALTERATION DEVICE 4056
ISOLATION DEVICE 4058
OPTIMISATION DEVICE 4048
MARKING DEVICE 4050
OPTIMISATION DEVICE 4046
COMPILER 4042
COMPILER MANAGER 4044
EXECUTION HISTORY RECORDER 4074
INTRERPRETER 4072
METHOD CHECKER 4054
CLASS LOADER 4052
OUTLIERS 4070
GLUE CODE 4068
EXECUTION DEVICE 4066

FIG. 9D

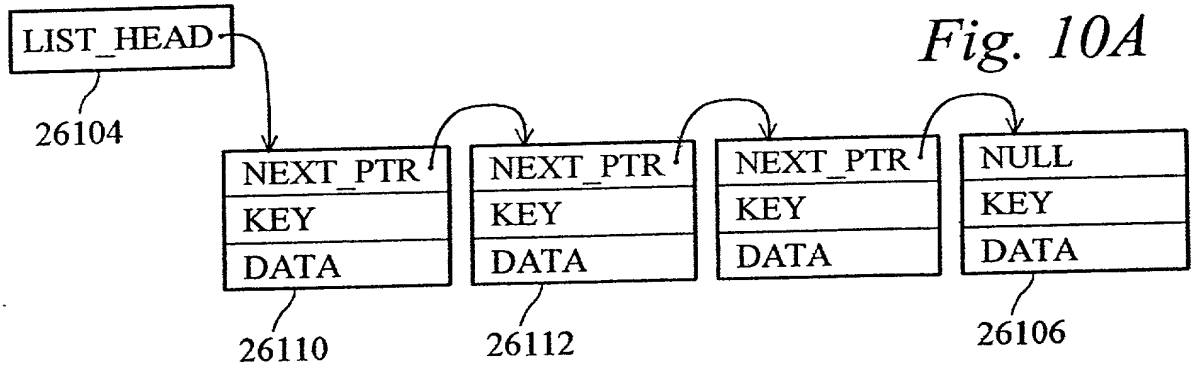


Fig. 10A

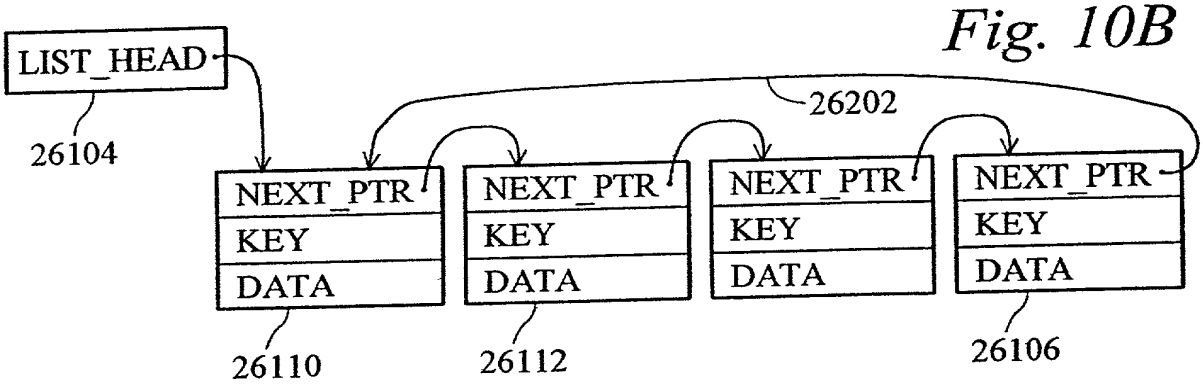


Fig. 10B

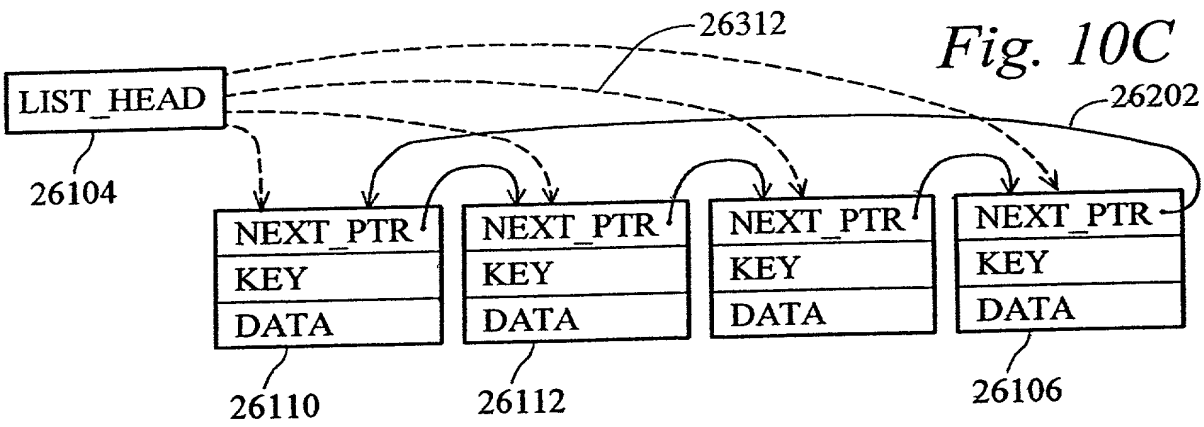


Fig. 10C

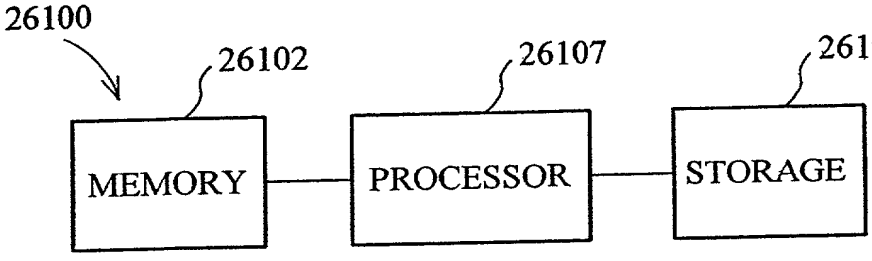


Fig. 10D

Fig. 11A

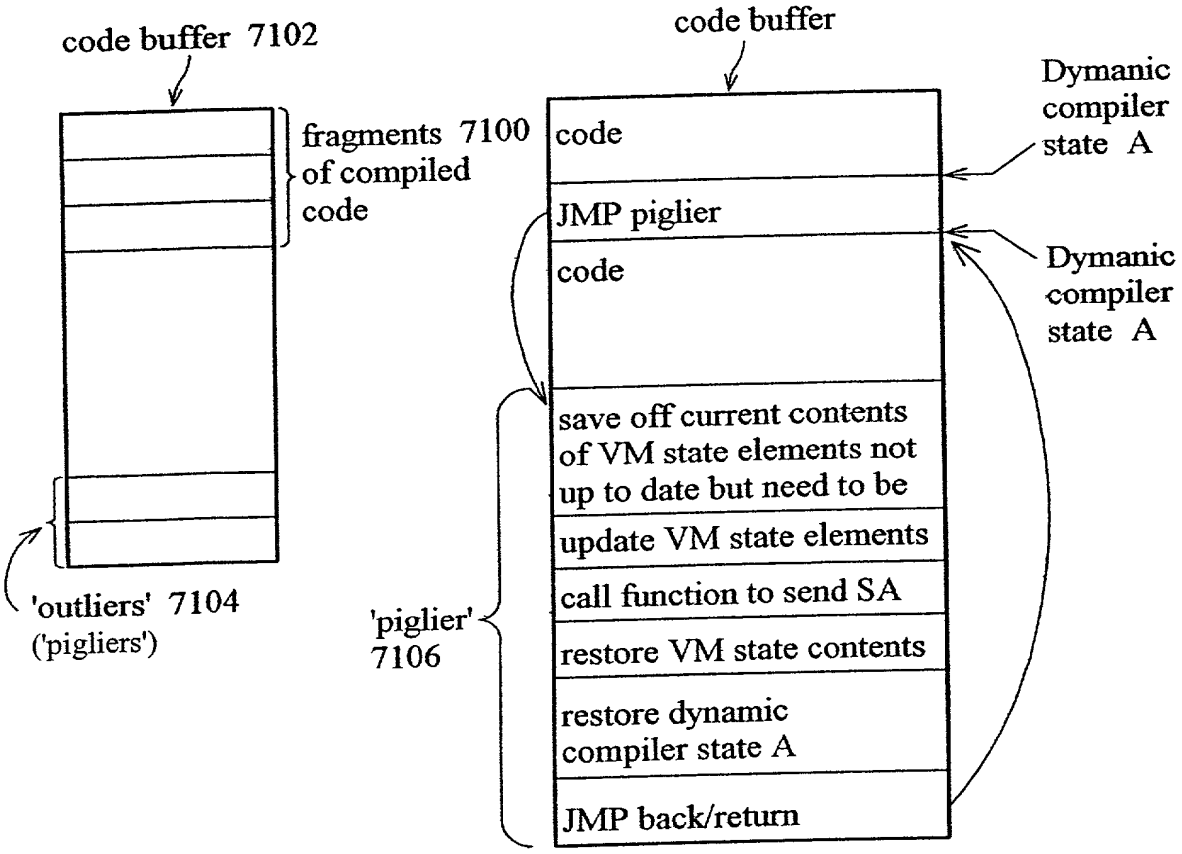
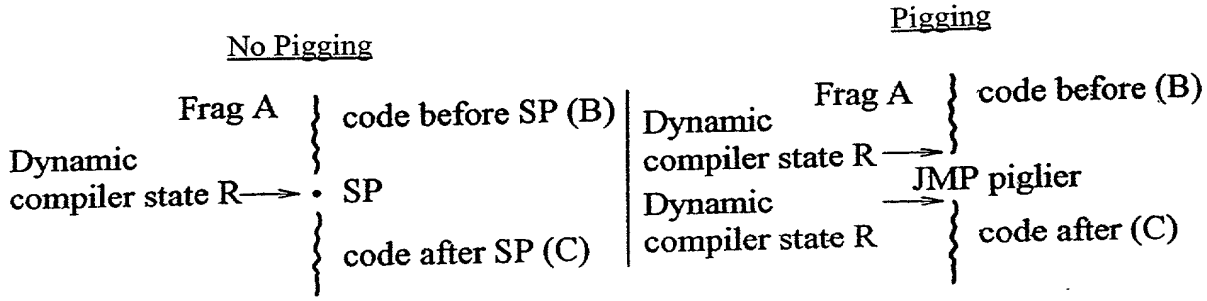


Fig. 11B



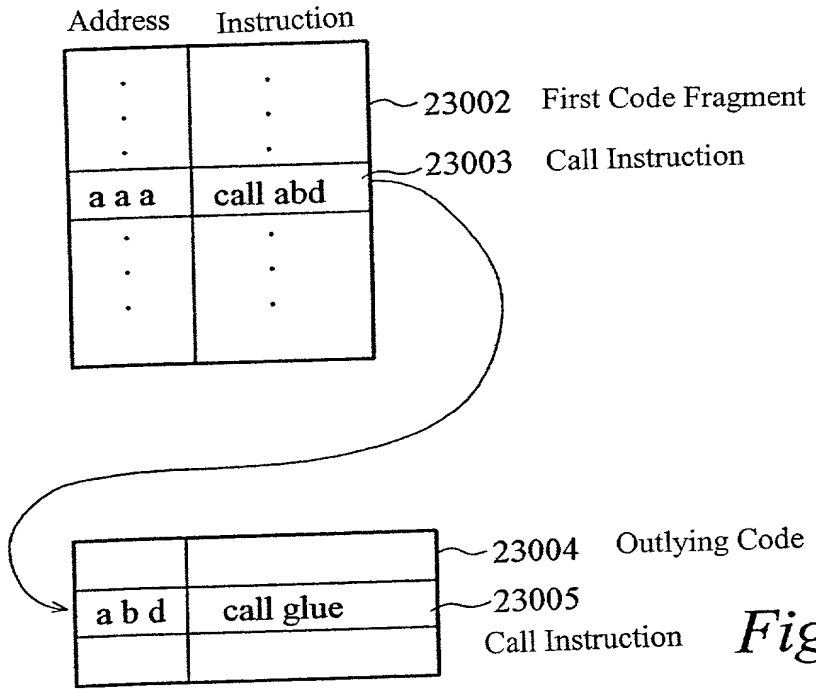


Fig. 12A

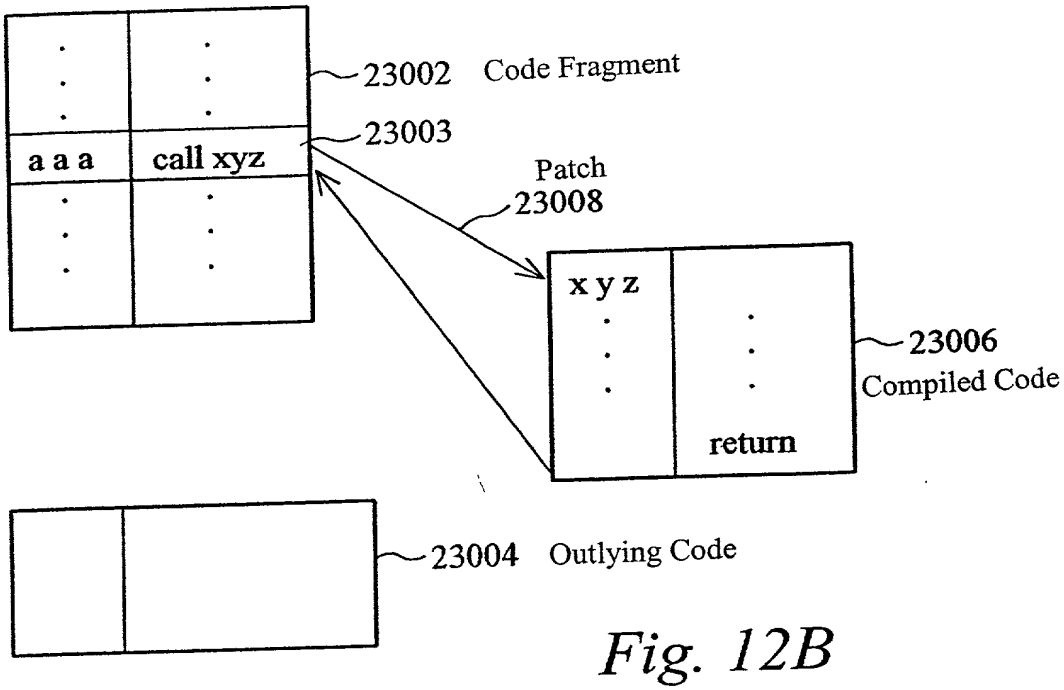


Fig. 12B

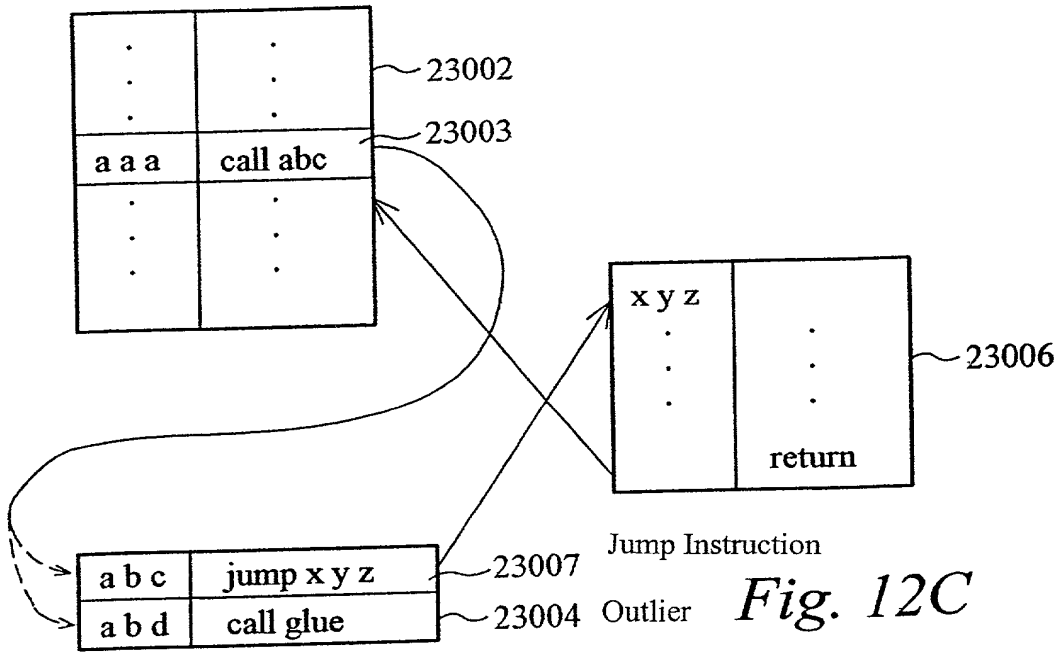


Fig. 12C

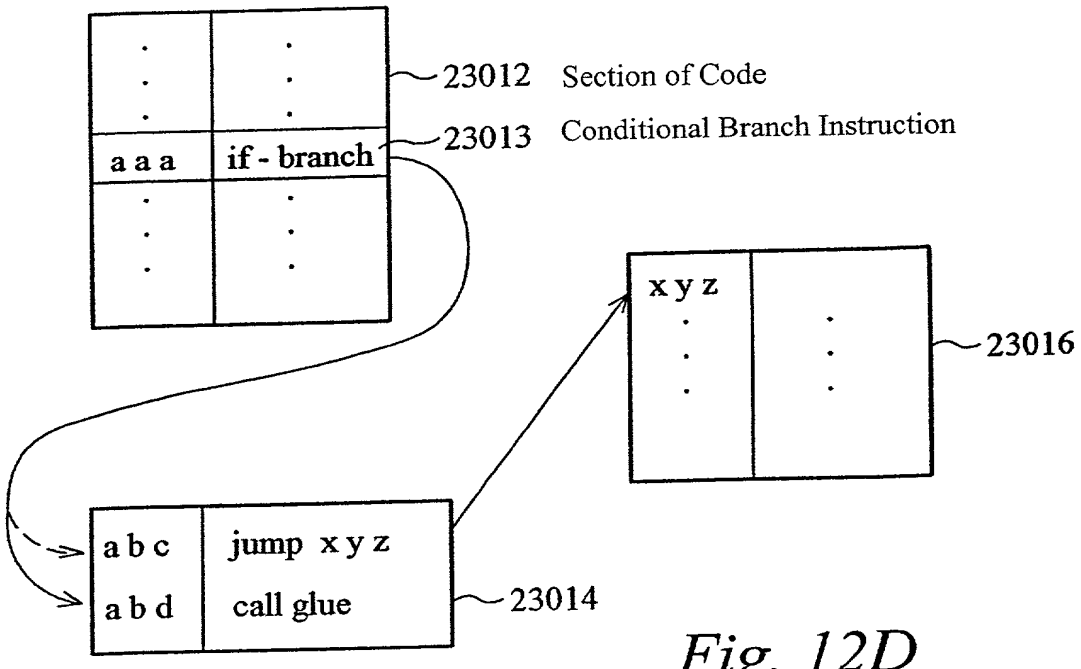


Fig. 12D

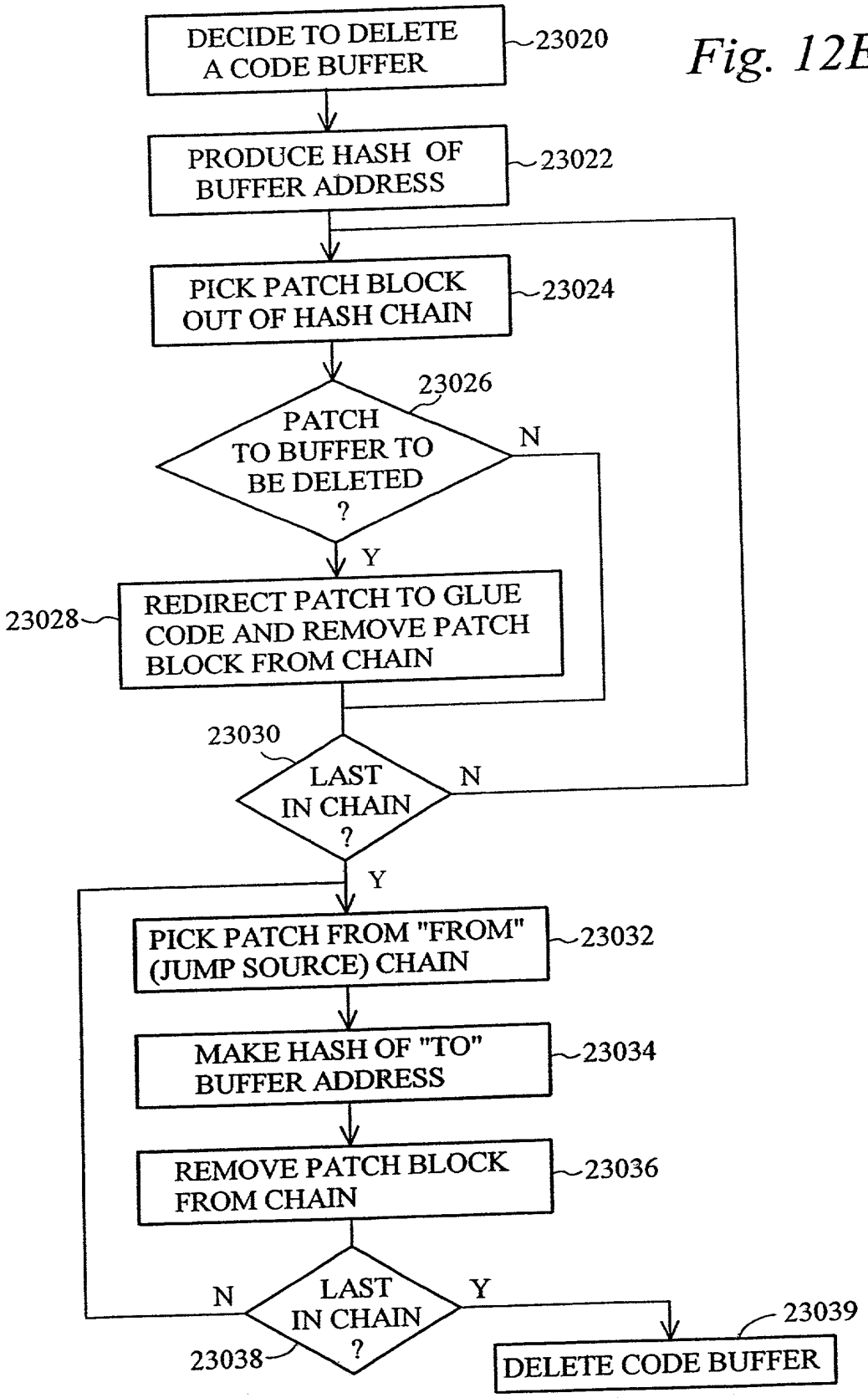

DECIDE TO DELETE A CODE BUFFER
23020
PRODUCE HASH OF BUFFER ADDRESS
23022
PICK PATCH BLOCK OUT OF HASH CHAIN
23024
23026
PATCH TO BUFFER TO BE DELETED ?
N
Y
23028
REDIRECT PATCH TO GLUE CODE AND REMOVE PATCH BLOCK FROM CHAIN
23030
LAST IN CHAIN ?
N
Y
PICK PATCH FROM "FROM" (JUMP SOURCE) CHAIN
23032
MAKE HASH OF "TO" BUFFER ADDRESS
23034
REMOVE PATCH BLOCK FROM CHAIN
23036
LAST IN CHAIN ?
N
Y
23038
23039
DELETE CODE BUFFER

Fig. 12E

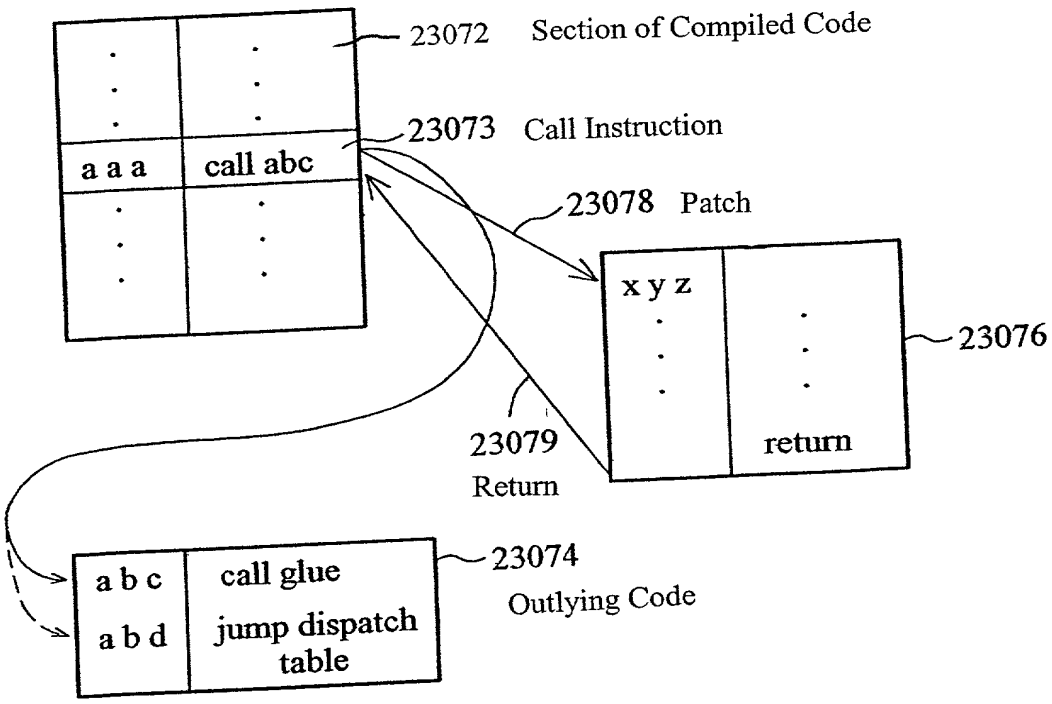

23072 Section of Compiled Code
23073 Call Instruction
a a a
call abc
23078 Patch
x y z
23076
return
23079
Return
a b c
call glue
a b d
jump dispatch table
23074
Outlying Code

Fig. 12F

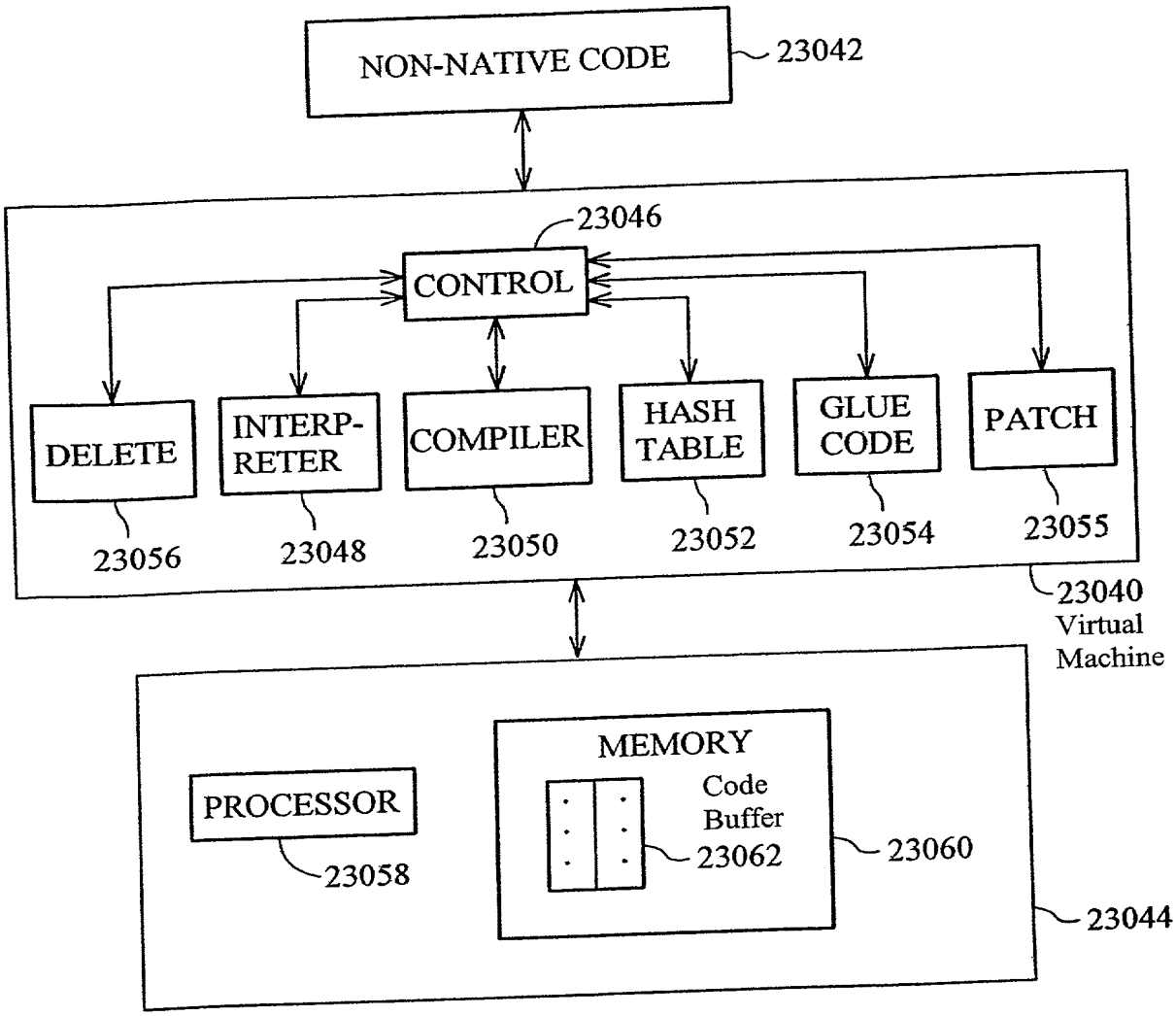

NON-NATIVE CODE
23042
23046
CONTROL
DELETE
INTERP-RETER
COMPILER
HASH TABLE
GLUE CODE
PATCH
23056
23048
23050
23052
23054
23055
23040
Virtual Machine
PROCESSOR
23058
MEMORY
Code Buffer
23062
23060
23044

Fig. 12G